



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

## Working Draft 02

~~13 March~~ 28 July 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)

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

Formatted: Portuguese (Brazil)

#### 42 Related work:

43 This specification is related to:

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

#### 46 Declared XML Namespace(s):

47 WS-HumanTask namespaces (defined in this specification):

- 48 • **htd** – <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803>
- 49 • ~~htdp~~ – <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/protocol/200803>
- 50 • **htdahta** – <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/services/200803><http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/200803>
- 51 • **htdtht** – <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/api/types/200803><http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/200803>
- 52 • **htc** – <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/context/200803>
- 53 • **htcp** – <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/protocol/200803>
- 54 • **htp** – <http://docs.oasis-open.org/ns/bpel4people/ws-humantask/policy/200803>

Field Code Changed

Formatted: English (United Kingdom)

Formatted: Bullets and Numbering

55 Other namespaces:

- 56 • **wsa** – <http://www.w3.org/2005/08/addressing>
- 57 • **wsdl** – <http://schemas.xmlsoap.org/wsdl/>
- 58 • **wsp** – <http://www.w3.org/ns/ws-policy>
- 59 • **xsd** – <http://www.w3.org/2001/XMLSchema>

#### 65 Abstract:

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

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

#### 75 Status:

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

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

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

87 The non-normative errata page for this specification is located at [http://www.oasis-  
open.org/committees/bpel4people/](http://www.oasis-<br/>88 open.org/committees/bpel4people/).

## Notices

90 Copyright © OASIS® 2008. All Rights Reserved.

91 All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual  
92 Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

93 This document and translations of it may be copied and furnished to others, and derivative works that  
94 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published,  
95 and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice  
96 and this section are included on all such copies and derivative works. However, this document itself may  
97 not be modified in any way, including by removing the copyright notice or references to OASIS, except as  
98 needed for the purpose of developing any document or deliverable produced by an OASIS Technical  
99 Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must  
100 be followed) or as required to translate it into languages other than English.

101 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors  
102 or assigns.

103 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
104 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY  
105 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY  
106 OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
107 PARTICULAR PURPOSE.

108 OASIS requests that any OASIS Party or any other party that believes it has patent claims that would  
109 necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard,  
110 to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to  
111 such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that  
112 produced this specification.

113 OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of  
114 any patent claims that would necessarily be infringed by implementations of this specification by a patent  
115 holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR  
116 Mode of the OASIS Technical Committee that produced this specification. OASIS may include such  
117 claims on its website, but disclaims any obligation to do so.

118 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that  
119 might be claimed to pertain to the implementation or use of the technology described in this document or  
120 the extent to which any license under such rights might or might not be available; neither does it  
121 represent that it has made any effort to identify any such rights. Information on OASIS' procedures with  
122 respect to rights in any document or deliverable produced by an OASIS Technical Committee can be  
123 found on the OASIS website. Copies of claims of rights made available for publication and any  
124 assurances of licenses to be made available, or the result of an attempt made to obtain a general license  
125 or permission for the use of such proprietary rights by implementers or users of this OASIS Committee  
126 Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no  
127 representation that any information or list of intellectual property rights will at any time be complete, or  
128 that any claims in such list are, in fact, Essential Claims.

129 The names "OASIS", [insert specific trademarked names and abbreviations here] are trademarks of  
130 OASIS, the owner and developer of this specification, and should be used only to refer to the organization  
131 and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications,  
132 while reserving the right to enforce its marks against misleading uses. Please see [http://www.oasis-  
open.org/who/trademark.php](http://www.oasis-<br/>133 open.org/who/trademark.php) for above guidance.

## Table of Contents

135	1	Introduction.....	<u>76</u>
136	2	Language Design .....	<u>87</u>
137	2.1	Dependencies on Other Specifications .....	<u>87</u>
138	2.2	Notational Conventions.....	<u>87</u>
139	2.3	Language Extensibility.....	<u>87</u>
140	2.4	Overall Language Structure.....	<u>98</u>
141	2.4.1	Syntax.....	<u>98</u>
142	2.4.2	Properties .....	<u>98</u>
143	3	Concepts .....	<u>1244</u>
144	3.1	Generic Human Roles .....	<u>1244</u>
145	3.2	Assigning People .....	<u>1342</u>
146	3.2.1	Using Logical People Groups.....	<u>1443</u>
147	3.2.2	Using Literals.....	<u>1544</u>
148	3.2.3	Using Expressions.....	<u>1544</u>
149	3.2.4	Data Type for Organizational Entities .....	<u>1646</u>
150	3.3	Task Rendering .....	<u>1746</u>
151	3.4	Task Instance Data.....	<u>1746</u>
152	3.4.1	Presentation Data.....	<u>1746</u>
153	3.4.2	Context Data.....	<u>1817</u>
154	3.4.3	Operational Data .....	<u>1817</u>
155	3.4.4	Data Types for Task Instance Data.....	<u>1948</u>
156	4	Human Tasks .....	<u>2322</u>
157	4.1	Overall Syntax .....	<u>2322</u>
158	4.2	Properties.....	<u>2423</u>
159	4.3	Presentation Elements.....	<u>2524</u>
160	4.4	Elements for Rendering Tasks .....	<u>2726</u>
161	4.5	Elements for People Assignment .....	<u>2827</u>
162	4.6	Elements for Handling Timeouts and Escalations.....	<u>2928</u>
163	4.7	Human Task Behavior and State Transitions.....	<u>3534</u>
164	4.7.1	Normal processing of a Human Task.....	<u>3635</u>
165	4.7.2	Releasing a Human Task.....	<u>3736</u>
166	4.7.3	Delegating or forwarding a Human Task.....	<u>3736</u>
167	4.7.4	Suspending and resuming a Human Task.....	<u>3736</u>
168	4.7.5	Skipping a Human Task .....	<u>3736</u>
169	4.7.6	Termination of a Human Task .....	<u>3736</u>
170	4.7.7	Error handling for Human Task .....	<u>3837</u>
171	5	Notifications .....	<u>3938</u>
172	5.1	Overall Syntax .....	<u>3938</u>
173	5.2	Properties.....	<u>4039</u>
174	5.3	Notification Behavior and State Transitions.....	<u>4039</u>
175	6	Programming Interfaces .....	<u>4140</u>
176	6.1	Operations for Client Applications .....	<u>4140</u>
177	6.1.1	Participant Operations.....	<u>4244</u>

178	6.1.2 Simple Query Operations .....	4746
179	6.1.3 Advanced Query Operation .....	4948
180	6.1.4 Administrative Operations .....	5254
181	6.2 XPath Extension Functions .....	5352
182	7 Interoperable Protocol for Advanced Interaction with Human Tasks .....	5756
183	7.1 Human Task Coordination Protocol Messages .....	5958
184	7.2 Protocol Messages .....	6059
185	7.2.1 Protocol Messages Received by a Task Parent .....	6059
186	7.2.2 Protocol Messages Received by a Task .....	6059
187	7.3 WSDL of the Protocol Endpoints .....	6159
188	7.3.1 Protocol Endpoint of the Task Parent .....	6160
189	7.3.2 Protocol Endpoint of the Task .....	6160
190	7.4 Providing Human Task Context .....	6160
191	7.4.1 Schema of the Human Task Context .....	6160
192	7.4.2 SOAP Binding of Human Task Context .....	6264
193	7.5 Human Task Policy Assertion .....	6463
194	8 Providing Callback Information for Human Tasks .....	6564
195	8.1 EPR Information Model Extension .....	6564
196	8.2 XML Infoset Representation .....	6564
197	8.3 Message Addressing Properties .....	6867
198	8.4 SOAP Binding .....	6867
199	9 Security Considerations .....	7274
200	10 Conformance .....	7372
201	11 References .....	7473
202	A. Portability and Interoperability Considerations .....	7675
203	B. WS-HumanTask Language Schema .....	7776
204	C. WS-HumanTask Data Types Schema .....	7877
205	D. WS-HumanTask API Port Types .....	7978
206	E. WS-HumanTask Protocol Handler Port Types .....	8079
207	F. WS-HumanTask Context Schema .....	8180
208	G. WS-HumanTask Policy Assertion Schema .....	8284
209	H. Sample .....	8382
210	I. Acknowledgements .....	8783
211	J. Non-Normative Text .....	8985
212	K. Revision History .....	9086
213		

214

## 1 Introduction

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

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

225 A human task has people assigned to it. These assignments define who should be allowed to  
226 play a certain role on that task. Human tasks may also specify how task metadata should be  
227 rendered on different devices or applications making them portable and interoperable with  
228 different types of software. Human tasks can be defined to react on timeouts, triggering an  
229 appropriate escalation action.

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

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

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

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

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

---

## 255 2 Language Design

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

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

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

### 269 2.1 Dependencies on Other Specifications

270 WS-HumanTask utilizes the following specifications:

- 271 • WSDL 1.1
- 272 • XML Schema 1.0
- 273 • XPath 1.0
- 274 • WS-Addressing 1.0
- 275 • WS-Coordination 1.1
- 276 • WS-Policy 1.5

### 277 2.2 Notational Conventions

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

### 281 2.3 Language Extensibility

282 The WS-HumanTask extensibility mechanism allows:

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

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

288 The specification differentiates between mandatory and optional extensions (the section below  
289 explains the syntax used to declare extensions). If a mandatory extension is used, a compliant  
290 implementation must understand the extension. If an optional extension is used, a compliant  
291 implementation may ignore the extension.

## 292 2.4 Overall Language Structure

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

### 296 2.4.1 Syntax

```
297 <htd:humanInteractions
298   xmlns:htd="http://www.example.org/ws-htd" http://docs.oasis-
299   open.org/ns/bpel4people/ws-humantask/200803"
300   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
301   xmlns:tns="anyURI"
302   targetNamespace="anyURI"
303   expressionLanguage="anyURI"?
304   queryLanguage="anyURI"?>
305
306   <htd:extensions>?
307     <htd:extension namespace="anyURI" mustUnderstand="yes|no"/>+
308   </htd:extensions>
309
310   <htd:import namespace="anyURI"?
311     location="anyURI"?
312     importType="anyURI" /*
313
314   <htd:logicalPeopleGroups>?
315     <htd:logicalPeopleGroup name="NCName" reference="QName"?>+
316       <htd:parameter name="NCName" type="QName" /*
317     </htd:logicalPeopleGroup>
318   </htd:logicalPeopleGroups>
319
320   <htd:tasks>?
321     <htd:task name="NCName">+
322       ...
323     </htd:task>
324   </htd:tasks>
325
326   <htd:notifications>?
327     <htd:notification name="NCName">+
328       ...
329     </htd:notification>
330   </htd:notifications>
331 </htd:humanInteractions>
332
333
```

### 334 2.4.2 Properties

335 The <humanInteractions> element has the following properties:

- 336 • `expressionLanguage`: This attribute specifies the expression language used in the  
337 enclosing elements. The default value for this attribute is `urn:ws-  
338 ht:sublang:xpath1.0` which represents the usage of XPath 1.0 within WS-  
339 HumanTask. The WS-HumanTask constructs that use expressions may override the  
340 default expression language for individual expressions. A WS-HumanTask compliant  
341 implementation MUST support the use of XPath 1.0 as the expression language.

- 342 • `queryLanguage`: This attribute specifies the query language used in the enclosing  
343 elements. The default value for this attribute is `urn:ws-ht:sublang:xpath1.0` which  
344 represents the usage of XPath 1.0 within WS-HumanTask. The WS-HumanTask  
345 constructs that use query expressions may override the default query language for  
346 individual query expressions. A WS-HumanTask compliant implementation MUST  
347 support the use of XPath 1.0 as the query language.
- 348 • `extensions`: This element is used to specify namespaces of WS-HumanTask extension  
349 attributes and extension elements. The element is optional. If present, it MUST include at  
350 least one extension element. The `<extension>` element is used to specify a namespace of  
351 WS-HumanTask extension attributes and extension elements, and indicate whether they  
352 are mandatory or optional. Attribute `mustUnderstand` is used to specify whether the  
353 extension must be understood by a compliant implementation. If the attribute has value  
354 "yes" the extension is mandatory. Otherwise, the extension is optional. If a WS-  
355 HumanTask implementation does not support one or more of the extensions with  
356 `mustUnderstand="yes"`, then the human interactions definition MUST be rejected.  
357 Optional extensions MAY be ignored. It is not required to declare optional extension. The  
358 same extension URI MAY be declared multiple times in the `<extensions>` element. If an  
359 extension URI is identified as mandatory in one `<extension>` element and optional in  
360 another, then the mandatory semantics have precedence and MUST be enforced. The  
361 extension declarations in an `<extensions>` element MUST be treated as an unordered  
362 set.
- 363 • `import`: This element is used to declare a dependency on external WS-HumanTask and  
364 WSDL definitions. Any number of `<import>` elements may appear as children of the  
365 `<humanInteractions>` element.
- 366 The `namespace` attribute specifies an absolute URI that identifies the imported  
367 definitions. This attribute is optional. An `<import>` element without a `namespace`  
368 attribute indicates that external definitions are in use which are not namespace-qualified.  
369 If a namespace is specified then the imported definitions MUST be in that namespace. If  
370 no namespace is specified then the imported definitions MUST NOT contain a  
371 `targetNamespace` specification. The `namespace`  
372 `http://www.w3.org/2001/XMLSchema` is imported implicitly. Note, however, that  
373 there is no implicit XML Namespace prefix defined for  
374 `http://www.w3.org/2001/XMLSchema`.
- 375 The `location` attribute contains a URI indicating the location of a document that  
376 contains relevant definitions. The `location` URI may be a relative URI, following the  
377 usual rules for resolution of the URI base [XML Base, RFC 2396]. The `location`  
378 attribute is optional. An `<import>` element without a `location` attribute indicates that  
379 external definitions are used by the process but makes no statement about where those  
380 definitions may be found. The `location` attribute is a hint and a WS-HumanTask  
381 compliant implementation is not required to retrieve the document being imported from  
382 the specified location.
- 383 The mandatory `importType` attribute identifies the type of document being imported by  
384 providing an absolute URI that identifies the encoding language used in the document.  
385 The value of the `importType` attribute MUST be set to  
386 <http://www.example.org/WS-HTTP>[http://docs.oasis-](http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803)  
387 [open.org/ns/bpel4people/ws-humantask/200803](http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803) when importing WS-  
388 HumanTask documents, or to <http://schemas.xmlsoap.org/wSDL/> when  
389 importing WSDL 1.1 documents.

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

395 A human interactions definition MUST import all WS-HumanTask and WSDL definitions it  
396 uses. In order to support the use of definitions from namespaces spanning multiple  
397 documents, a human interactions definition MAY include more than one import  
398 declaration for the same namespace and importType, provided that those declarations  
399 include different location values. <import> elements are conceptually unordered. A  
400 human interactions definition MUST be rejected if the imported documents contain  
401 conflicting definitions of a component used by the importing process definition.

402 Documents (or namespaces) imported by an imported document (or namespace) MUST  
403 NOT be transitively imported by a WS-HumanTask compliant implementation. In  
404 particular, this means that if an external item is used by a task enclosed in the human  
405 interactions definition, then a document (or namespace) that defines that item MUST be  
406 directly imported by the human interactions definition. This requirement does not limit the  
407 ability of the imported document itself to import other documents or namespaces.

- 408 • logicalPeopleGroups: This element specifies a set of all logical people groups used  
409 in the enclosing human tasks and notifications. The element is optional. If present, it  
410 MUST include at least one logicalPeopleGroup element. The set of logical people groups  
411 MUST contain only those logical people groups that are used in the humanInteractions  
412 element, and enclosed human tasks and notifications. The logicalPeopleGroup element  
413 has the following attributes. The name attribute specifies the name of the logical people  
414 group. The name MUST be unique among the names of all logicalPeopleGroups defined  
415 within the humanInteractions element. The reference attribute is optional. In the reference  
416 attribute specifies logical people group, in case a logical people group is used in the  
417 humanInteractions element that is defined in an elsewhere imported WS-HumanTask  
418 definition, the reference attribute MUST be used to specify the logical people group. The  
419 reference attribute is optional. The parameter element is used to pass data needed for  
420 people query evaluation.
- 421 • tasks: This element specifies a set of human tasks. The element is optional. If present, it  
422 MUST include at least one <task> element. The syntax and semantics of the <task>  
423 element are introduced in section 4 "Human Tasks".
- 424 • notifications: This element specifies a set of notifications. The element is optional. If  
425 present, it MUST include at least one <notification> element. The syntax and semantics  
426 of the <notification> element are introduced in section 5 "Notifications".

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

429  
430 All WS-HumanTask elements may use the element <documentation> to provide annotation for  
431 users. The content could be a plain text, HTML, and so on. The <documentation> element is  
432 optional and has the following syntax:

```
433  
434 <htd:documentation xml:lang="xsd:language">  
435 ...  
436 </htd:documentation>
```

Formatted: French (Canada)

437

## 3 Concepts

438

### 3.1 Generic Human Roles

439

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:

440

441

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

442

443

444

445

446

447

448

449

450

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.

451

452

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.

453

454

455

456

457

458

*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.

459

460

461

462

463

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

464

465

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.

466

467

468

469

*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.

470

471

472

473

*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.

474

475

476

477

## 478 3.2 Assigning People

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

- 482 • Via logical people groups (see 3.2.1 “Using Logical People Groups”)
- 483 • Via literals (see 3.2.2 “Using Literals”)
- 484 • Via expressions e.g., by retrieving data from the input message of the human task (see  
485 3.2.3 “Using Expressions”).

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

### 489 Syntax:

```
490 <htd:peopleAssignments>  
491  
492   <htd:genericHumanRole>+  
493     <htd:from>...</htd:from>  
494   </htd:genericHumanRole>  
495  
496 </htd:peopleAssignments>
```

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

```
500 <htd:potentialOwners>  
501   <htd:from>...</htd:from>  
502 </htd:potentialOwners>  
503  
504 <htd:excludedOwners>  
505   <htd:from>...</htd:from>  
506 </htd:excludedOwners>  
507  
508 <htd:taskInitiator>  
509   <htd:from>...</htd:from>  
510 </htd:taskInitiator>  
511  
512 <htd:taskStakeholders>  
513   <htd:from>...</htd:from>  
514 </htd:taskStakeholders>  
515  
516 <htd:businessAdministrators>  
517   <htd:from>...</htd:from>  
518 </htd:businessAdministrators>  
519  
520 <htd:recipients>  
521   <htd:from>...</htd:from>  
522 </htd:recipients>
```

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

### 525 3.2.1 Using Logical People Groups

526 A *logical people group* represents either one person, a set of people, or one or many unresolved  
527 groups of people (i.e., group names). A logical people group is bound to a people query against a  
528 people directory at deployment time. Though the term *query* is used, the exact discovery and  
529 invocation mechanism of this query is not defined by this specification. There are no limitations as  
530 to how the logical people group is evaluated. At runtime, this people query is evaluated to retrieve  
531 the actual people assigned to the task or notification. Logical people groups support query  
532 parameters which are passed to the people query at runtime. Parameters may refer to task  
533 instance data (see section 3.4 for more details). During people query execution an infrastructure  
534 may decide which of the parameters defined by the logical people group are used. It may use  
535 zero or more of the parameters specified. It may also override certain parameters with values  
536 defined during logical people group deployment. The deployment mechanism for tasks and  
537 logical people groups is out of scope for this specification.

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

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

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

557

#### 558 **Syntax:**

```
559 <htd:from logicalPeopleGroup="NCName">  
560   <htd:argument name="NCName" expressionLanguage="anyURI"? > *  
561     expression  
562   </htd:argument>  
563 </htd:from>
```

564

565 The `logicalPeopleGroup` attribute refers to a `logicalPeopleGroup` definition. The element  
566 `<argument>` is used to pass values used in the people query. The `expressionLanguage`  
567 attribute specifies the language used in the expression. The attribute is optional. If not specified,  
568 the default language as inherited from the closest enclosing element that specifies the attribute is  
569 used.

570

#### 571 **Example:**

```
572 <htd:potentialOwners>  
573   <htd:from logicalPeopleGroup="regionalClerks">  
574     <htd:argument name="region">  
575       htd:getInput("part1")/region  
576     </htd:argument>
```

```
577 </htd:from>
578 </htd:potentialOwners>
```

### 579 3.2.2 Using Literals

580 People assignments can be defined literally by directly specifying the user identifier(s) or the  
581 name(s) of groups using either the `htd:tOrganizationalEntity` or `htd:tUser` data type  
582 introduced below (see 3.2.4 "Data Type for Organizational Entities").

#### 583 Syntax:

```
584 <htd:from>
585   <htd:literal>
586     ... literal value ...
587   </htd:literal>
588 </htd:from>
```

589

#### 590 Example specifying user identifiers:

```
591 <htd:potentialOwners>
592   <htd:from>
593     <htd:literal>
594       <htd:organizationalEntity>
595         <htd:users>
596           <htd:user>Alan</htd:user>
597           <htd:user>Dieter</htd:user>
598           <htd:user>Frank</htd:user>
599           <htd:user>Gerhard</htd:user>
600           <htd:user>Ivana</htd:user>
601           <htd:user>Karsten</htd:user>
602           <htd:user>Matthias</htd:user>
603           <htd:user>Patrick</htd:user>
604         </htd:users>
605       </htd:organizationalEntity>
606     </htd:literal>
607   </htd:from>
608 </htd:potentialOwners>
609
```

#### 610 Example specifying group names:

```
611 <htd:potentialOwners>
612   <htd:from>
613     <htd:literal>
614       <htd:organizationalEntity>
615         <htd:groups>
616           <htd:group>bpel4people authors</htd:group>
617         </htd:groups>
618       </htd:organizationalEntity>
619     </htd:literal>
620   </htd:from>
621 </htd:potentialOwners>
```

### 622 3.2.3 Using Expressions

623 Alternatively people can be assigned using expressions returning either an instance of the  
624 `htd:tOrganizationalEntity` data type or the `htd:tUser` data type introduced below (see  
625 3.2.4 "Data Type for Organizational Entities").

626

627 **Syntax:**

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

631  
632 The `expressionLanguage` attribute specifies the language used in the expression. The  
633 attribute is optional. If not specified, the default language as inherited from the closest enclosing  
634 element that specifies the attribute is used.

635

636 **Example:**

```
637 <htd:potentialOwners>
638   <htd:from>htd:getInput("part1")/approvers</htd:from>
639 </htd:potentialOwners>
640
641 <htd:businessAdministrators>
642   <htd:from>
643     htd:except(_htd:getInput("part1")/admins,
644               _htd:getInput("part1")/globaladmins[0]_)
645   </htd:from>
646 </htd:businessAdministrators>
```

### 647 3.2.4 Data Type for Organizational Entities

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

```
652 <xsd:element name="organizationalEntity" type="tOrganizationalEntity" />
653 <xsd:complexType name="tOrganizationalEntity">
654   <xsd:choice>
655     <xsd:element ref="users" />
656     <xsd:element ref="groups" />
657   </xsd:choice>
658 </xsd:complexType>
659
660 <xsd:element name="user" type="tUser" />
661 <xsd:simpleType name="tUser">
662   <xsd:restriction base="xsd:string" />
663 </xsd:simpleType>
664
665 <xsd:element name="users" type="tUserlist" />
666 <xsd:complexType name="tUserlist">
667   <xsd:sequence>
668     <xsd:element ref="user" minOccurs="0" maxOccurs="unbounded" />
669   </xsd:sequence>
670 </xsd:complexType>
671
672 <xsd:element name="group" type="tGroup" />
673 <xsd:simpleType name="tGroup">
674   <xsd:restriction base="xsd:string" />
675 </xsd:simpleType>
676
677 <xsd:element name="groups" type="tGrouplist" />
678 <xsd:complexType name="tGrouplist">
679   <xsd:sequence>
```

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

### 683 3.3 Task Rendering

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

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

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

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

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

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

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

### 715 3.4 Task Instance Data

716 Task instance data falls into three categories:

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

#### 723 3.4.1 Presentation Data

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

### 727 3.4.2 Context Data

728 The task context includes the following:

- 729 • Task state
- 730 • Priority
- 731 • Values for all generic human roles, i.e. potential owners, actual owner and business  
732 administrators
- 733 • Time stamps such as start time, completion time, [defer expiration time](#), and expiration  
734 time
- 735 • Skipable indicator

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

### 739 3.4.3 Operational Data

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

#### 744 3.4.3.1 Ad-hoc Attachments

745 Arbitrary additional data may be attached to a task. This additional data is referred to as *task ad-*  
746 *hoc attachments*. An ad-hoc attachment is specified by its name, its type and its content.

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

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

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

758 The `attachedAt` element indicates when the attachment is added.

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

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

763

#### 764 Attachment Info Data Type

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

```
766 <xsd:element name="attachmentInfo" type="tAttachmentInfo" />  
767 <xsd:complexType name="tAttachmentInfo">  
768   <xsd:sequence>  
769     <xsd:element name="name" type="xsd:string" />  
770     <xsd:element name="accessType" type="xsd:string" />
```

```

771 <xsd:element name="contentType" type="xsd:string" />
772 <xsd:element name="attachedAt" type="xsd:dateTime" />
773 <xsd:element name="attachedBy" type="htd:tUser" />
774 <xsd:any namespace="##other" processContents="lax"
775 minOccurs="0" maxOccurs="unbounded" />
776 </xsd:sequence>
777 </xsd:complexType>
778

```

### 779 Attachment Data Type

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

```

781 <xsd:element name="attachment" type="tAttachment" />
782 <xsd:complexType name="tAttachment">
783 <xsd:sequence>
784 <xsd:element ref="attachmentInfo" />
785 <xsd:element name="value" type="xsd:anyType" />
786 </xsd:sequence>
787 </xsd:complexType>

```

### 788 3.4.3.2 Comments

789 A task may have associated textual notes added by participants of the task. These notes are  
790 collectively referred to as *task comments*. Comments are essentially a chronologically ordered list  
791 of notes added by various users who worked on the task. A comment has the text, user  
792 information and a timestamp. Comments are usually added individually, but retrieved as one  
793 group. Comments usage is optional in a task.

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

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

797

### 798 Comment Data Type

799 The following data type is used to return comments.

```

800 <xsd:element name="comment" type="tComment" />
801 <xsd:complexType name="tComment">
802 <xsd:sequence>
803 <xsd:element name="addedAt" type="xsd:dateTime" />
804 <xsd:element name="addedBy" type="htd:tUser" />
805 <xsd:element name="text" type="xsd:string" />
806 <xsd:any namespace="##other" processContents="lax"
807 minOccurs="0" maxOccurs="unbounded" />
808 </xsd:sequence>
809 </xsd:complexType>

```

810

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

### 812 3.4.4 Data Types for Task Instance Data

813 The following data types are used to represent instance data of a task or a notification. The data  
814 type `htd:taskAbstract`/`htt:tTaskAbstract` is used to provide the summary data of a task  
815 or a notification that is displayed on a task list. The data type `htd:task`/`htt:tTask` contains the  
816 data of a task or a notification, except ad-hoc attachments, comments and presentation  
817 description. The data that is not contained in `htd:task`/`htt:tTask` may be retrieved separately  
818 from the task engine using the task API.

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

822 The elements `startByExists` and `completeByExists` have a value of “true” if the task has  
823 at least one start deadline or at least one completion deadline respectively. The actual times  
824 (`startBy` and `completeBy`) of the individual deadlines can be retrieved using the query  
825 operation (see section 6.1.3 “Advanced Query Operation”).

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

827

## 828 TaskAbstract Data Type

```
829 <xsd:element name="taskAbstract" type="tTaskAbstract" />
830 <xsd:complexType name="tTaskAbstract">
831   <xsd:sequence>
832     <xsd:element name="id"
833       type="xsd:string" />
834     <xsd:element name="taskType"
835       type="xsd:string" />
836     <xsd:element name="name"
837       type="xsd:QName" />
838     <xsd:element name="status"
839       type="tStatus" />
840     <xsd:element name="priority"
841       type="xsd:nonNegativeInteger" minOccurs="0" />
842     <xsd:element name="createdOn"
843       type="xsd:dateTime" />
844     <xsd:element name="activationTime"
845       type="xsd:dateTime" minOccurs="0" />
846     <xsd:element name="expirationTime"
847       type="xsd:dateTime" minOccurs="0" />
848     <xsd:element name="isSkipable"
849       type="xsd:boolean" minOccurs="0" />
850     <xsd:element name="hasPotentialOwners"
851       type="xsd:boolean" minOccurs="0" />
852     <xsd:element name="startByExists"
853       type="xsd:boolean" minOccurs="0" />
854     <xsd:element name="completeByExists"
855       type="xsd:boolean" minOccurs="0" />
856     <xsd:element name="presentationName"
857       type="tPresentationName" minOccurs="0" />
858     <xsd:element name="presentationSubject"
859       type="tPresentationSubject" minOccurs="0" />
860     <xsd:element name="renderingMethodExists"
861       type="xsd:boolean" />
862     <xsd:element name="hasOutput"
863       type="xsd:boolean" minOccurs="0" />
864     <xsd:element name="hasFault"
865       type="xsd:boolean" minOccurs="0" />
866     <xsd:element name="hasAttachments"
867       type="xsd:boolean" minOccurs="0" />
868     <xsd:element name="hasComments"
869       type="xsd:boolean" minOccurs="0" />
870     <xsd:element name="escalated"
871       type="xsd:boolean" minOccurs="0" />
872     <xsd:any namespace="##other" processContents="lax"
873       minOccurs="0" maxOccurs="unbounded" />
```

```
874 </xsd:sequence>
875 </xsd:complexType>
```

876

### 877 Task Data Type

```
878 <xsd:element name="task" type="tTask"/>
879 <xsd:complexType name="tTask">
880   <xsd:sequence>
881     <xsd:element name="id"
882       type="xsd:string"/>
883     <xsd:element name="taskType"
884       type="xsd:string"/>
885     <xsd:element name="name"
886       type="xsd:QName"/>
887     <xsd:element name="status"
888       type="tStatus"/>
889     <xsd:element name="priority"
890       type="xsd:nonNegativeInteger" minOccurs="0"/>
891     <xsd:element name="taskInitiator"
892       type="htd:tUser" minOccurs="0"/>
893     <xsd:element name="taskStakeholders"
894       type="htd:tOrganizationalEntity" minOccurs="0"/>
895     <xsd:element name="potentialOwners"
896       type="htd:tOrganizationalEntity" minOccurs="0"/>
897     <xsd:element name="businessAdministrators"
898       type="htd:tOrganizationalEntity" minOccurs="0"/>
899     <xsd:element name="actualOwner"
900       type="htd:tUser" minOccurs="0"/>
901     <xsd:element name="notificationRecipients"
902       type="htd:tOrganizationalEntity" minOccurs="0"/>
903     <xsd:element name="createdOn"
904       type="xsd:dateTime"/>
905     <xsd:element name="createdBy"
906       type="xsd:string" minOccurs="0"/>
907     <xsd:element name="activationTime"
908       type="xsd:dateTime" minOccurs="0"/>
909     <xsd:element name="expirationTime"
910       type="xsd:dateTime" minOccurs="0"/>
911     <xsd:element name="isSkipable"
912       type="xsd:boolean" minOccurs="0"/>
913     <xsd:element name="hasPotentialOwners"
914       type="xsd:boolean" minOccurs="0"/>
915     <xsd:element name="startByExists"
916       type="xsd:boolean" minOccurs="0"/>
917     <xsd:element name="completeByExists"
918       type="xsd:boolean" minOccurs="0"/>
919     <xsd:element name="presentationName"
920       type="tPresentationName" minOccurs="0"/>
921     <xsd:element name="presentationSubject"
922       type="tPresentationSubject" minOccurs="0"/>
923     <xsd:element name="renderingMethodExists"
924       type="xsd:boolean"/>
925     <xsd:element name="hasOutput"
926       type="xsd:boolean" minOccurs="0"/>
927     <xsd:element name="hasFault"
928       type="xsd:boolean" minOccurs="0"/>
929     <xsd:element name="hasAttachments"
```

```

930         type="xsd:boolean" minOccurs="0"/>
931     <xsd:element name="hasComments"
932         type="xsd:boolean" minOccurs="0"/>
933     <xsd:element name="escalated"
934         type="xsd:boolean" minOccurs="0"/>
935     <xsd:element name="primarySearchBy"
936         type="xsd:string" minOccurs="0"/>
937     <xsd:any namespace="##other" processContents="lax"
938         minOccurs="0" maxOccurs="unbounded"/>
939 </xsd:sequence>
940 </xsd:complexType>

```

941

## 942 Common Data Types

```

943 <xsd:simpleType name="tPresentationName">
944     <xsd:annotation>
945         <xsd:documentation>length-restricted string</xsd:documentation>
946     </xsd:annotation>
947     <xsd:restriction base="xsd:string">
948         <xsd:maxLength value="64" />
949         <xsd:whiteSpace value="preserve" />
950     </xsd:restriction>
951 </xsd:simpleType>
952
953 <xsd:simpleType name="tPresentationSubject">
954     <xsd:annotation>
955         <xsd:documentation>length-restricted string</xsd:documentation>
956     </xsd:annotation>
957     <xsd:restriction base="xsd:string">
958         <xsd:maxLength value="254" />
959         <xsd:whiteSpace value="preserve" />
960     </xsd:restriction>
961 </xsd:simpleType>
962
963 <xsd:simpleType name="tStatus">
964     <xsd:restriction base="xsd:string">
965         <xsd:enumeration value="CREATED" />
966         <xsd:enumeration value="READY" />
967         <xsd:enumeration value="RESERVED" />
968         <xsd:enumeration value="IN_PROGRESS" />
969         <xsd:enumeration value="SUSPENDED" />
970         <xsd:enumeration value="COMPLETED" />
971         <xsd:enumeration value="FAILED" />
972         <xsd:enumeration value="ERROR" />
973         <xsd:enumeration value="EXITED" />
974         <xsd:enumeration value="OBSOLETE" />
975     </xsd:restriction>
976 </xsd:simpleType>

```

977

## 4 Human Tasks

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

### 4.1 Overall Syntax

981 Definition of human tasks:

```
982 <htd:task name="NCName">
983
984   <htd:interface portType="QName" operation="NCName"
985     responsePortType="QName"? responseOperation="NCName"? />
986
987   <htd:priority expressionLanguage="anyURI"? >?
988     integer-expression
989   </htd:priority>
990
991   <htd:peopleAssignments>...</htd:peopleAssignments>
992
993   <htd:delegation
994     potentialDelegates="anybody|nobody|potentialOwners|other" />?
995     <htd:from?
996       ...
997     </htd:from>
998   </htd:delegation>
999
1000   <htd:presentationElements>...</htd:presentationElements>
1001
1002   <htd:outcome part="NCName" queryLanguage="anyURI"?>?
1003     queryContent
1004   </htd:outcome>
1005
1006   <htd:searchBy expressionLanguage="anyURI"? >?
1007     expression
1008   </htd:searchBy>
1009
1010   <htd:renderings?
1011     <htd:rendering type="QName">+
1012     ...
1013   </htd:rendering>
1014 </htd:renderings>
1015
1016   <htd:deadlines?
1017     <htd:startDeadline>*
1018     ...
1019   </htd:startDeadline>
1020
1021     <htd:completionDeadline>*
1022     ...
1023   </htd:completionDeadline>
1024 </htd:deadlines>
1025
1026 </htd:task>
```

1027  
1028 </htd:task>

## 1029 4.2 Properties

1030 The following attributes and elements are defined for tasks:

- 1031 • **name**: This attribute is used to specify the name of the task. The name combined with the  
1032 target namespace of a task element is used to uniquely identify the task definition. This  
1033 attribute is mandatory. It is not used for task rendering.
- 1034 • **interface**: This element is used to specify the operation used to invoke the task. The  
1035 operation is specified using WSDL, that is, a WSDL port type and WSDL operation are  
1036 defined. The element and its `portType` and `operation` attributes are mandatory. The  
1037 interface is specified in one of the following forms:
  - 1038 ▪ The WSDL operation is a **one-way** operation and the task  
1039 asynchronously returns output data. In this case, a callback one-way  
1040 operation **MUST** be specified, using the `responsePortType` and  
1041 `responseOperation` attributes. This callback operation is invoked  
1042 when the task has finished. The Web service endpoint address of the  
1043 callback operation is provided at runtime when the task's one-way  
1044 operation is invoked (for details, see section 8 "Providing Callback  
1045 Information for Human Tasks").
  - 1046 ▪ The WSDL operation is a **request-response** operation. In this case, the  
1047 `responsePortType` and `responseOperation` attributes **MUST NOT**  
1048 be specified.
- 1049 • **priority**: This element is used to specify the priority of the task. It is an optional  
1050 element which value is an integer expression. If not present, the priority of the task is  
1051 unspecified. 0 is the highest priority, larger numbers identify lower priorities. The result of  
1052 the expression evaluation is of type `xsd:integer`. The `expressionLanguage`  
1053 attribute specifies the language used in the expression. The attribute is optional. If not  
1054 specified, the default language as inherited from the closest enclosing element that  
1055 specifies the attribute is used.
- 1056 • **peopleAssignments**: This element is used to specify people assigned to different  
1057 generic human roles, i.e. potential owners, and business administrator. The element is  
1058 mandatory. See section 0 for more details on people assignments.
- 1059 • **delegation**: This element is used to specify constraints concerning delegation of the  
1060 task. Attribute `potentialDelegates` defines to whom the task may be delegated. The  
1061 following values are allowed:
  - 1062 ▪ **anybody**: It is allowed to delegate the task to anybody
  - 1063 ▪ **potentialOwners**: It is allowed to delegate the task to potential  
1064 owners previously selected
  - 1065 ▪ **other**: It is allowed to delegate the task to other people, e.g. authorized  
1066 owners. The element `<from>` is used to determine the people to whom  
1067 the task may be delegated.
  - 1068 ▪ **nobody**: It is not allowed to delegate the task.

1069 The delegation element is optional. If this element is not present the task is allowed to be  
1070 delegated to anybody.

- 1071 • `presentationElements`: This element is used to specify different information used to  
1072 display the task in a task list, such as name, subject and description. See section 4.3 for  
1073 more details on presentation elements. The element is mandatory.
- 1074 • `outcome`: This optional element identifies the field (of an `xsd:string` type) in the output  
1075 message which reflects the business result of the task. A conversion takes place to yield  
1076 an outcome of type `xsd:string`. The optional attribute `queryLanguage` specifies the  
1077 language used for selection. If not specified, the default language as inherited from the  
1078 closest enclosing element that specifies the attribute is used.
- 1079 • `searchBy`: This optional element is used to search for task instances based on a custom  
1080 search criterion. The result of the expression evaluation is of type `xsd:string`. The  
1081 `expressionLanguage` attribute specifies the language used in the expression. The  
1082 attribute is optional. If not specified, the default language as inherited from the closest  
1083 enclosing element that specifies the attribute is used.
- 1084 • `rendering`: This element is used to specify the rendering method. It is optional. If not  
1085 present, task rendering is implementation dependent. See section 4.4 for more details on  
1086 rendering tasks.
- 1087 • `deadlines`: This element specifies different deadlines. It is optional. See section 4.6 for  
1088 more details on timeouts and escalations.

### 1089 4.3 Presentation Elements

1090 Information about human tasks or notifications needs to be made available in a human-readable  
1091 way to allow users dealing with their tasks and notifications via a user interface, which could be  
1092 based on various technologies, such as Web browsers, Java clients, Flex-based clients or .NET  
1093 clients. For example, a user queries for her tasks, getting a list of tasks she should work on,  
1094 displaying a short description of each task. Upon selection of one of the tasks, more complete  
1095 information about the task is displayed by the user interface.

1096 Alternatively, a task or notification could be sent directly to a user's inbox, in which case the same  
1097 information would be used to provide a human readable rendering there.

1098 The same human readable information could also be used in reports on all the human tasks  
1099 executed by a particular human task management system.

1100 Human readable information may be specified in multiple languages.

1101

#### 1102 Syntax:

```
1103 <htd:presentationElements>
1104
1105   <htd:name xml:lang="xsd:language"? >*<br>
1106     Text<br>
1107   </htd:name>
1108
1109   <!-- For the subject and description only,<br>
1110     replacement variables can be used. -->
1111   <htd:presentationParameters expressionLanguage="anyURI"? >?<br>
1112     <htd:presentationParameter name="NCName" type="QName">+<br>
1113       expression<br>
1114     </htd:presentationParameter>
1115   </htd:presentationParameters>
1116
1117   <htd:subject xml:lang="xsd:language"? >*<br>
1118     Text<br>
1119   </htd:subject>
```

```

1120
1121     <htd:description xml:lang="xsd:language"?
1122     contentType="mimeTypeString"? >*
1123     <xsd:any minOccurs="0" maxOccurs="unbounded" />
1124     </htd:description>
1125
1126 </htd:presentationElements>
1127

```

## 1128 Properties

1129 The following attributes and elements are defined for the `htd:presentationElements`  
1130 element.

- 1131 • `name`: This element is the short title of a task. It uses `xml:lang`, a standard XML  
1132 attribute, to define the language of the enclosed information. This attribute uses tags  
1133 according to RFC 1766 (see [RFC1766]). There could be zero or more `name` elements. It  
1134 is not allowed to specify multiple `name` elements having the same value for attribute  
1135 `xml:lang`.
- 1136 • `presentationParameters`: This element specifies parameters used in presentation  
1137 elements `subject` and `description`. Attribute `expressionLanguage` identifies  
1138 the expression language used to define parameters. This attribute is optional. If not  
1139 specified, the default language as inherited from the closest enclosing element that  
1140 specifies the attribute is used. Element `presentationParameters` is optional and if  
1141 present MUST specify at least one element `presentationParameter`. Element  
1142 `presentationParameter` has attribute `name`, which uniquely identifies the parameter  
1143 definition within the `presentationParameters` element, and attribute `type` which  
1144 defines its type. Parameters MUST be of XSD simple types. When a  
1145 `presentationParameter` is used within `subject` and `description`, the syntax is  
1146 `{$parameterName}`. The pair `"{"` represents the character `"{"` and the pair `"}"`  
1147 represents the character `"}"`. Only the defined presentation parameters and not arbitrary  
1148 expressions are allowed to be embedded with this syntax.
- 1149 • `subject`: This element is a longer text that describes the task. It uses `xml:lang` to  
1150 define the language of the enclosed information. There could be zero or more `subject`  
1151 elements. It is not allowed to specify multiple `subject` elements having the same value  
1152 for attribute `xml:lang`.
- 1153 • `description`: This element is a long description of the task. It uses `xml:lang` to  
1154 define the language of the enclosed information. The optional attribute `contentType`  
1155 uses content types according to RFC 2046 (see [RFC 2046]). The default value for this  
1156 attribute is `"text/plain"`. A compliant implementation MUST support the content type  
1157 `"text/plain"`. It SHOULD support HTML (such as `"text/html"` or `"application/xml+xhtml"`).  
1158 There could be zero or more `description` elements. As descriptions may exist with  
1159 different content types, it is allowed to specify multiple `description` elements having  
1160 the same value for attribute `xml:lang`, but their content types MUST be different.

1161

## 1162 Example:

```

1163 <htd:presentationElements>
1164
1165     <htd:name xml:lang="en-US">Approve Claim</htd:name>
1166     <htd:name xml:lang="de-DE">
1167         Genehmigung der Schadensforderung
1168     </htd:name>

```

```

1169
1170 <htd:presentationParameters>
1171   <htd:presentationParameter name="firstname" type="xsd:string">
1172     htd:getInput("ClaimApprovalRequest")/cust/firstname
1173   </htd:presentationParameter>
1174   <htd:presentationParameter name="lastname" type="xsd:string">
1175     htd:getInput("ClaimApprovalRequest")/cust/lastname
1176   </htd:presentationParameter>
1177   <htd:presentationParameter name="euroAmount" type="xsd:double">
1178     htd:getInput("ClaimApprovalRequest")/amount
1179   </htd:presentationParameter>
1180 </htd:presentationParameters>
1181
1182 <htd:subject xml:lang="en-US">
1183   Approve the insurance claim for €{$euroAmount} on behalf of
1184   {$firstname} {$lastname}
1185 </htd:subject>
1186 <htd:subject xml:lang="de-DE">
1187   Genehmigung der Schadensforderung über €{$euroAmount} für
1188   {$firstname} {$lastname}
1189 </htd:subject>
1190
1191 <htd:description xml:lang="en-US" contentType="text/plain">
1192   Approve this claim following corporate guideline #4711.0815/7 ...
1193 </htd:description>
1194 <htd:description xml:lang="en-US" contentType="text/html">
1195   <p>
1196     Approve this claim following corporate guideline
1197     <b>#4711.0815/7</b>
1198     ...
1199   </p>
1200 </htd:description>
1201 <htd:description xml:lang="de-DE" contentType="text/plain">
1202   Genehmigen Sie diese Schadensforderung entsprechend Richtlinie Nr.
1203   4711.0815/7 ...
1204 </htd:description>
1205 <htd:description xml:lang="de-DE" contentType="text/html">
1206   <p>
1207     Genehmigen Sie diese Schadensforderung entsprechend Richtlinie
1208     <b>Nr. 4711.0815/7</b>
1209     ...
1210   </p>
1211 </htd:description>
1212
1213 </htd:presentationElements>
1214

```

Formatted: French (France)

Formatted: French (France)

Formatted: German (Germany)

#### 1215 4.4 Elements for Rendering Tasks

1216 Human tasks and notifications need to be rendered on user interfaces like forms clients, portlets,  
1217 e-mail clients, etc. The rendering element provides an extensible mechanism for specifying UI  
1218 renderings for human tasks and notifications (task-UI). The element is optional. One or more  
1219 rendering methods may be provided in a task definition or a notification definition. A task or  
1220 notification can be deployed on any compliant implementation, irrespective of the fact whether the  
1221 implementation supports specified rendering methods or not. The rendering method is identified  
1222 using a QName.

1223 Unlike for presentation elements, language considerations are opaque for the rendering element  
1224 because the rendering applications typically provide multi-language support. Where this is not the  
1225 case, providers of certain rendering types may decide to extend the rendering method in order to  
1226 provide language information for a given rendering.

1227 The content of the rendering element is not defined by this specification. For example, when used  
1228 in the rendering element, XPath extension functions as defined in section 6.2 may or may not be  
1229 evaluated by a compliant implementation.

1230

1231 **Syntax:**

```
1232 <htd:renderings>  
1233   <htd:rendering type="QName">+  
1234     <xsd:any minOccurs="1" maxOccurs="1" />  
1235   </htd:rendering>  
1236 </htd:renderings>
```

## 1237 4.5 Elements for People Assignment

1238 The <peopleAssignments> element is used to assign people to the task. For each generic  
1239 human role, a people assignment element can be specified. For human tasks it is mandatory to  
1240 specify people assignment for potential owners. If no potential owner should be assigned by the  
1241 human task's definition, e.g. because nomination is used, then this is accomplished by adding an  
1242 empty <potentialOwners> element. Specifying people assignments for task stakeholders,  
1243 task initiators, excluded owners and business administrators is optional. Human tasks never  
1244 specify recipients. People assignments for actual owners MUST NOT be specified.

1245

1246 **Syntax:**

```
1247 <htd:peopleAssignments>  
1248  
1249   <htd:potentialOwners>  
1250     ...  
1251   </htd:potentialOwners>  
1252  
1253   <htd:excludedOwners?>  
1254     ...  
1255   </htd:excludedOwners?>  
1256  
1257   <htd:taskInitiator?>  
1258     ...  
1259   </htd:taskInitiator?>  
1260  
1261   <htd:taskStakeholders?>  
1262     ...  
1263   </htd:taskStakeholders?>  
1264  
1265   <htd:businessAdministrators?>  
1266     ...  
1267   </htd:businessAdministrators?>  
1268  
1269 </htd:peopleAssignments>
```

1270

1271 People assignments may result in a set of values or an empty set. In case people assignment  
1272 results in an empty set then the task may require administrative attention. This is out of scope of

1273 the specification, except for people assignments for potential owners (see section 4.7.1 "Normal  
1274 processing of a Human Task" for more details).

1275

1276 **Example:**

```
1277 <htd:peopleAssignments>  
1278   <htd:potentialOwners>  
1279     <htd:from logicalPeopleGroup="regionalClerks">  
1280       <htd:argument name="region">  
1281         htd:getInput ("ClaimApprovalRequest")/region  
1282       </htd:argument>  
1283     </htd:from>  
1284   </htd:potentialOwners>  
1285  
1286   <htd:businessAdministrators>  
1287     <htd:from logicalPeopleGroup="regionalManager">  
1288       <htd:argument name="region">  
1289         htd:getInput ("ClaimApprovalRequest")/region  
1290       </htd:argument>  
1291     </htd:from>  
1292   </htd:businessAdministrators>  
1293 </htd:peopleAssignments>
```

## 1294 4.6 Elements for Handling Timeouts and Escalations

1295 Timeouts and escalations allow the specification of a date or time before which the task must  
1296 reach a specific state. If the timeout occurs a set of actions is performed as the response. The  
1297 state of the task is not changed. Several deadlines are specified which differ in the point when the  
1298 timer clock starts and the state which must be reached with the given duration or by the given  
1299 date. They are:

- 1300 • **Start deadline:** Specifies the time until the task must start, i.e. it must reach state  
1301 *InProgress*. It is defined as either the period of time or the point in time until the task must  
1302 reach state *InProgress*. Since expressions are allowed, durations and deadlines can be  
1303 calculated at runtime, which for example enables custom calendar integration. The time  
1304 starts to be measured from the time at which the task enters the state *Created*. If the task  
1305 does not reach state *InProgress* by the deadline an escalation action or a set of  
1306 escalation actions is performed. Once the task is started, the timer becomes obsolete.
- 1307 • **Completion deadline:** Specifies the due time of the task. It is defined as either the period  
1308 of time until the task gets due or the point in time when the task gets due. The time starts  
1309 to be measured from the time at which the task enters the state *Created*. If the task does  
1310 not reach one of the final states (*Completed*, *Failed*, *Error*, *Exited*, *Obsolete*) by the  
1311 deadline an escalation action or a set of escalation actions is performed.

1312 The element `<deadlines>` is used to include the definition of all deadlines within the task  
1313 definition. It is optional. If present, at least one deadline MUST be defined.

1314

1315 **Syntax:**

```
1316 <htd:deadlines>  
1317   <htd:startDeadline>*</htd:startDeadline>  
1318  
1319   <htd:documentation xml:lang="xsd:language"? >*</htd:documentation>  
1320     Text  
1321   </htd:documentation>  
1322 </htd:deadlines>
```

Formatted: French (France)

```

1323
1324 ( <htd:for expressionLanguage="anyURI"? >
1325     duration-expression
1326 </htd:for>
1327 | <htd:until expressionLanguage="anyURI"? >
1328     deadline-expression
1329 </htd:until>
1330 )
1331
1332 <htd:escalation name="NCName"*
1333     ...
1334 </htd:escalation>
1335
1336 </htd:startDeadline>
1337
1338 <htd:completionDeadline>*
1339     ...
1340 </htd:completionDeadline>
1341
1342 </htd:deadlines>

```

1343

1344 The language used in expressions is specified using the `expressionLanguage` attribute. This  
 1345 attribute is optional. If not specified, the default language as inherited from the closest enclosing  
 1346 element that specifies the attribute is used.

1347 For all deadlines if a status is not reached within a certain time then an escalation action,  
 1348 specified using element `<escalation>`, can be triggered. The `<escalation>` element is  
 1349 defined in the section below. When the task reaches a final state (*Completed, Failed, Error,*  
 1350 *Exited, Obsolete*) all deadlines are deleted.

1351

1352 Escalations are triggered if

- 1353 1. The associated point in time is reached, or duration has elapsed, and
- 1354 2. The associated condition (if any) evaluates to true

1355 Escalations use notifications to inform people about the status of the task. Optionally, a task  
 1356 might be reassigned to some other person or group as part of the escalation. Notifications are  
 1357 explained in more detail in section 5 "Notifications". An escalation MUST specify exactly one  
 1358 escalation action.

1359 When defining escalations, a notification can be either referred to, or defined inline.

- 1360 • A notification defined in the `<humanInteractions>` root element or imported from a  
 1361 different namespace can be referenced by specifying its QName in the `reference`  
 1362 attribute of a `<localNotification>` element. When referring to a notification, the  
 1363 priority and the people assignments of the original notification definition MAY be  
 1364 overridden using the elements `<priority>` and `<peopleAssignments>` contained in  
 1365 the `<localNotification>` element.
- 1366 • A inlined notification is defined by a `<notification>` element.

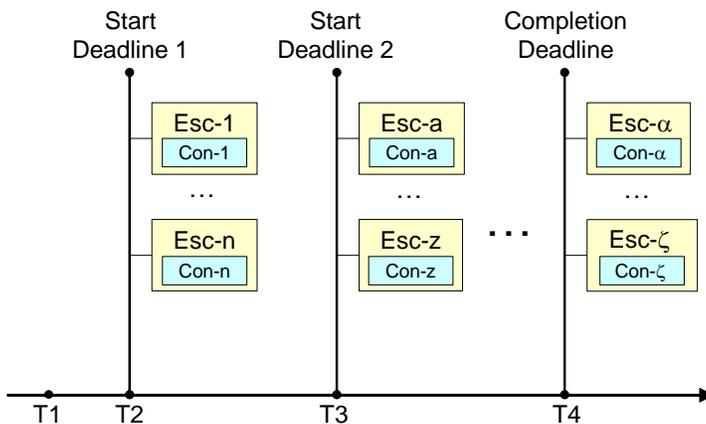
1367 Notifications used in escalations may use the same type of input data as the surrounding task, or  
 1368 different type of data. If the same type of data is used then the input message of the task is  
 1369 passed to the notification implicitly. If not, then the `<toPart>` elements are used to assign  
 1370 appropriate data to the notification, i.e. to explicitly create a multi-part WSDL message from the  
 1371 data. The `part` attribute refers to a part of the WSDL message. The `expressionLanguage`  
 1372 attribute specifies the language used in the expression. The attribute is optional. If not specified,

1373 the default language as inherited from the closest enclosing element that specifies the attribute is  
1374 used.

1375 There MUST be a <toPart> element for every part in the WSDL message definition because  
1376 parts not explicitly represented by <toPart> elements would result in uninitialized parts in the  
1377 target WSDL message. The order in which parts are specified is not relevant. If multiple  
1378 <toPart> elements are present, they MUST be executed in an "all or nothing" manner. If any of  
1379 the <toPart>s fails, the escalation action will not be performed and the execution of the task is not  
1380 affected.

1381 Reassignments are used to replace the potential owners of a task when an escalation is  
1382 triggered. The <reassignment> element is used to specify reassignment. If present, the  
1383 element MUST specify potential owners.

1384 In the case where several reassignment escalations are triggered, the first reassignment (lexical  
1385 order) will be considered for execution. The task is set to state *Ready* after reassignment.  
1386 Reassignments and notifications are performed in the lexical order.



1387  
1388 A task may have multiple start deadlines and completion deadlines associated with it. Each such  
1389 deadline encompasses escalation actions each of which may send notifications to certain people.  
1390 The corresponding set of people may overlap.

1391 As an example, the figure depicts a task that has been created at time T1. Its two start deadlines  
1392 would be missed at time T2 and T3, respectively. The associated escalations whose conditions  
1393 evaluate to "true" are triggered. Both, the escalations Esc-1 to Esc-n as well as escalations Esc-a  
1394 to Esc-z may involve an overlapping set of people. The completion deadline would be missed at  
1395 time T4.

1396

1397 **Syntax:**

```
1398 <htd:deadlines>  
1399   <htd:startDeadline>*  
1400   ...  
1401   <htd:escalation name="NCName">*  
1402   ...  
1403   <htd:condition expressionLanguage="anyURI"?>?  
1404  
1405
```

```

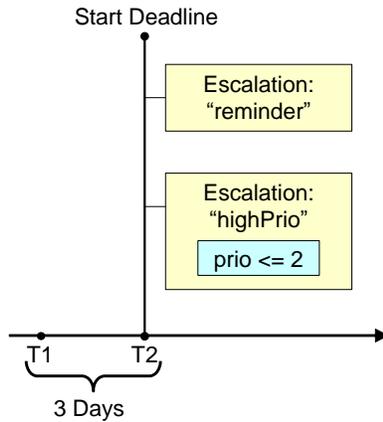
1406     boolean-expression
1407 </htd:condition>
1408
1409 <htd:toParts>?
1410   <htd:toPart part="NCName"
1411     expressionLanguage="anyURI"?>+
1412     expression
1413   </htd:toPart>
1414 </htd:toParts>
1415
1416 <!-- notification specified by reference -->
1417 <htd:localNotification reference="QName"?
1418   <htd:priority expressionLanguage="anyURI"?>?
1419   integer-expression
1420 </htd:priority>
1421 <htd:peopleAssignments>?
1422   <htd:recipients>
1423     ...
1424   </htd:recipients>
1425 </htd:peopleAssignments>
1426
1427 </htd:localNotification>
1428
1429 <!-- notification specified inline -->
1430 <htd:notification name="NCName"?
1431   ...
1432 </htd:notification>
1433
1434 <htd:reassignment>?
1435
1436   <htd:potentialOwners>
1437     ...
1438   </htd:potentialOwners>
1439
1440 </htd:reassignment>
1441
1442 </htd:escalation>
1443
1444 </htd:startDeadline>
1445
1446 <htd:completionDeadline>*
1447   ...
1448 </htd:completionDeadline>
1449
1450 </htd:deadlines>

```

1451

**Example:**

1453 The following example shows the specification of a start deadline with escalations. At runtime, the  
 1454 following picture depicts the result of what is specified in the example:



1455 The human task is created at T1. If it has not been started, i.e., no person is working on it until  
 1456 T2, then the escalation "reminder" is triggered that notifies the potential owners of the task that  
 1457 work is waiting for them. In case the task has high priority then at the same time the regional  
 1458 manager is informed. If the task amount is greater than or equal 10000 the task is reassigned to  
 1459 Alan.  
 1460 In case that task has been started before T2 was reached, then the start deadline is deactivated,  
 1461 no escalation occurs.

1462

```

1463 <htd:startDeadline>
1464   <htd:documentation xml:lang="en-US">
1465     If not started within 3 days, - escalation notifications are sent
1466     if the claimed amount is less than 10000 - to the task's potential
1467     owners to remind them or their todo - to the regional manager, if
1468     this approval is of high priority (0,1, or 2) - the task is
1469     reassigned to Alan if the claimed amount is greater than or equal
1470     10000
1471   </htd:documentation>
1472   <htd:for>P3D</htd:for>
1473
1474   <htd:escalation name="reminder">
1475
1476     <htd:condition>
1477       <![CDATA[
1478         htd:getInput("ClaimApprovalRequest")/amount < 10000
1479       ]]>
1480     </htd:condition>
1481
1482     <htd:toParts>
1483       <htd:toPart name="firstname">
1484         htd:getInput("ClaimApprovalRequest","ApproveClaim") /firstname
1485       </htd:toPart>
1486       <htd:toPart name="lastname">
1487         htd:getInput("ClaimApprovalRequest","ApproveClaim") /lastname
1488       </htd:toPart>
1489       <htd:toPart name="taskId">
1490         htd:getTaskID("ApproveClaim")
  
```

```

1491 | </htd:toPart>
1492 | </htd:toParts>
1493 |
1494 | <htd:localNotification reference="tns:ClaimApprovalReminder">
1495 |
1496 |   <htd:documentation xml:lang="en-US">
1497 |     Reuse the predefined notification "ClaimApprovalReminder".
1498 |     Overwrite the recipients with the task's potential owners.
1499 |   </htd:documentation>
1500 |
1501 |   <htd:peopleAssignments>
1502 |     <htd:recipients>
1503 |       <htd:from>htd:getPotentialOwners("ApproveClaim")</htd:from>
1504 |     </htd:recipients>
1505 |   </htd:peopleAssignments>
1506 |
1507 | </htd:localNotification>
1508 |
1509 | </htd:escalation>
1510 |
1511 | <htd:escalation name="highPrio">
1512 |
1513 |   <htd:condition>
1514 |     <![CDATA[
1515 |       (htd:getInput("ClaimApprovalRequest")/amount < 10000
1516 |         && htd:getInput("ClaimApprovalRequest")/prio <= 2)
1517 |     ]]>
1518 |   </htd:condition>
1519 |
1520 |   <!-- task input implicitly passed to the notification -->
1521 |
1522 |   <htd:notification name="ClaimApprovalOverdue">
1523 |     <htd:documentation xml:lang="en-US">
1524 |       An inline defined notification using the approval data as its
1525 |       input.
1526 |     </htd:documentation>
1527 |
1528 |     <htd:interface portType="tns:ClaimsHandlingPT"
1529 |       operation="escalate" />
1530 |
1531 |     <htd:peopleAssignments>
1532 |       <htd:recipients>
1533 |         <htd:from logicalPeopleGroup="regionalManager">
1534 |           <htd:argument name="region">
1535 |             htd:getInput("ClaimApprovalRequest")/region
1536 |           </htd:argument>
1537 |         </htd:from>
1538 |       </htd:recipients>
1539 |     </htd:peopleAssignments>
1540 |
1541 |     <htd:presentationElements>
1542 |       <htd:name xml:lang="en-US">Claim approval overdue</htd:name>
1543 |       <htd:name xml:lang="de-DE">
1544 |         Überfällige Schadensforderungsgenehmigung
1545 |       </htd:name>
1546 |     </htd:presentationElements>
1547 |

```

Formatted: German (Germany)

Formatted: German (Germany)

```

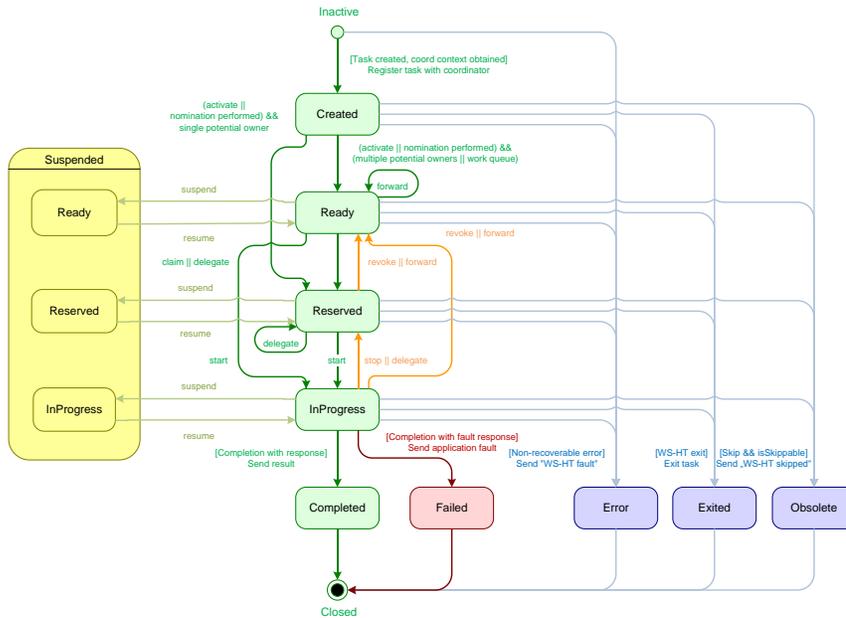
1548 | </htd:notification>
1549 |
1550 | </htd:escalation>
1551 |
1552 | <htd:escalation name="highAmountReassign">
1553 |
1554 |   <htd:condition>
1555 |     <![CDATA[
1556 |       htd:getInput("ClaimApprovalRequest")/amount >= 10000
1557 |     ]]>
1558 |   </htd:condition>
1559 |
1560 |   <htd:reassignment>
1561 |     <htd:documentation>
1562 |       Reassign task to Alan if amount is greater than or equal
1563 |       10000.
1564 |     </htd:documentation>
1565 |
1566 |     <htd:potentialOwners>
1567 |       <htd:from>
1568 |         <htd:literal>
1569 |           <htd:organizationalEntity>
1570 |             <htd:users>
1571 |               <htd:user>Alan</htd:user>
1572 |             </htd:users>
1573 |           </htd:organizationalEntity>
1574 |         </htd:literal>
1575 |       </htd:from>
1576 |     </htd:potentialOwners>
1577 |
1578 |   </htd:reassignment>
1579 |
1580 | </htd:escalation>
1581 |
1582 | </htd:startDeadline>

```

Formatted: German (Germany)

## 1583 4.7 Human Task Behavior and State Transitions

1584 Human tasks can have a number of different states and substates. The state diagram for human  
 1585 tasks below shows the different states and the transitions between them.



1586

#### 1587 4.7.1 Normal processing of a Human Task

1588 Upon creation, a task goes into its initial state *Created*. Task creation starts with the initialization  
 1589 of its properties in the following order:

- 1590 1. Input message
- 1591 2. Priority
- 1592 3. Generic human roles (such as excluded owners, potential owners and business  
 1593 administrators) are made available in the lexical order of their definition in the people  
 1594 assignment definition with the constraint that excluded owners are taken into account  
 1595 when evaluating the potential owners.
- 1596 4. All other properties are evaluated after these properties in an implementation dependent  
 1597 order.

1598 Task creation succeeds irrespective of whether the people assignment returns a set of values or  
 1599 an empty set. People queries that cannot be executed successfully are treated as if they were  
 1600 returning an empty set.

1601 If potential owners were not assigned automatically during task creation, they must be assigned  
 1602 explicitly using nomination, which is performed by the task's business administrator. The result of  
 1603 evaluating potential owners removes the excluded owners from results. The task remains in the  
 1604 state *Created* until it is activated (i.e., an activation timer has been specified) and has potential  
 1605 owners.

1606 When the task has a single potential owner, it transitions into the *Reserved* state, indicating that it  
 1607 is assigned to a single actual owner. Otherwise (i.e., when it has multiple potential owners or is  
 1608 assigned to a work queue), it transitions into the *Ready* state, indicating that it can be claimed by  
 1609 one of its potential owners. Once a potential owner claims the task, it transitions into the  
 1610 *Reserved* state, making that potential owner the actual owner.

1611 Once work is started on a task that is in state *Ready* or *Reserved*, it goes into the *InProgress*  
1612 state, indicating that it is being worked on – if the transition is from *Ready*, the user starting the  
1613 work becomes its actual owner.  
1614 On successful completion of the work, the task transitions into the *Completed* final state. On  
1615 unsuccessful completion of the work (i.e., with an exception), the task transitions into the *Failed*  
1616 final state.

#### 1617 **4.7.2 Releasing a Human Task**

1618 The current actual owner of a human task may *release* a task to again make it available for all  
1619 potential owners. A task can be released from active states that have an actual owner (*Reserved*,  
1620 *InProgress*), transitioning it into the *Ready* state. Business data associated with the task  
1621 (intermediate result data, ad-hoc attachments and comments) is kept.  
1622 A task that is currently *InProgress* can be stopped by the actual owner, transitioning it into state  
1623 *Reserved*. Business data associated with the task as well as its actual owner is kept.

#### 1624 **4.7.3 Delegating or forwarding a Human Task**

1625 Task's potential owners, actual owner or business administrator can *delegate* a task to another  
1626 user, making that user the actual owner of the task, and also adding her to the list of potential  
1627 owners in case she is not, yet. A task can be delegated when it is in an active state (*Ready*,  
1628 *Reserved*, *InProgress*), and transitions the task into the *Reserved* state. Business data  
1629 associated with the task is kept.  
1630 Similarly, task's potential owners, actual owner or business administrator can forward an active  
1631 task to another person or a set of people, replacing himself by those people in the list of potential  
1632 owners. Potential owners can only forward tasks that are in the *Ready* state. Forwarding is  
1633 possible if the task has a set of individually assigned potential owners, not if its potential owners  
1634 are assigned using one or many groups. If the task is in the *Reserved* or *InProgress* state then  
1635 the task is implicitly released first, that is, the task is transitioned into the *Ready* state. Business  
1636 data associated with the task is kept. The user performing the forward is removed from the set of  
1637 potential owners of the task, and the forwarder is added to the set of potential owners.

#### 1638 **4.7.4 Suspending and resuming a Human Task**

1639 In any of its active states (*Ready*, *Reserved*, *InProgress*), a task can be suspended, transitioning  
1640 it into the *Suspended* state. The *Suspended* state has sub-states to indicate the original state of  
1641 the task.  
1642 On resumption of the task, it transitions back to the original state from which it had been  
1643 suspended.

#### 1644 **4.7.5 Skipping a Human Task**

1645 A person working on a human task or a business administrator may decide that a task is no  
1646 longer needed, and hence skip this task. This transitions the task into the *Obsolete* state. This is  
1647 considered a "good" outcome of a task, even though an empty result is returned. The enclosing  
1648 environment can be notified of that transition as described in section 0.  
1649 The task can only be skipped if this capability is specified during the task invocation. A side-effect  
1650 of this is that a task which is invoked using basic Web service protocols is not skipable.

#### 1651 **4.7.6 Termination of a Human Task**

1652 The enclosing environment of a human task (such as the calling application or business process)  
1653 may decide that a task is no longer needed and terminate it, either because a timeout has  
1654 reached in that enclosing context (i.e., the task has expired), or because the enclosing  
1655 environment itself is terminated. These events transition the task into the *Obsolete* state.

1656 **4.7.7 Error handling for Human Task**

1657 If a human task encounters a non-recoverable error in any of its state (for example, it executes a  
1658 divide by zero in an XPath expression), it transitions into the *Error* state. This is considered a  
1659 “bad” outcome of the task and no result is returned. The enclosing environment can be notified of  
1660 that transition as described in section 0.

1661

## 5 Notifications

1662 Notifications are used to notify a person or a group of people of a noteworthy business event,  
1663 such as that a particular order has been approved, or a particular product is about to be shipped.  
1664 They are also used in escalation actions to notify a user that a task is overdue or a task has not  
1665 been started yet. The person or people to whom the notification will be assigned to could be  
1666 provided, for example, as result of a people query to organizational model.

1667 Notifications are simple human interactions that do not block the progress of the caller, that is the  
1668 caller does not wait for the notification to be completed. Moreover, the caller cannot influence the  
1669 execution of notifications, e.g. notifications are not terminated if the caller terminates. The caller,  
1670 i.e. an application, a business process or an escalation action, initiates a notification passing the  
1671 required notification data. The notification appears on the task list of all notification recipients.  
1672 After a notification recipient removes it, the notification disappears from the recipient's task list.

1673 A notification may have multiple recipients and optionally one or many business administrators.  
1674 The generic human roles task initiator, task stakeholders, potential owners, actual owner and  
1675 excluded owners play no role.

1676 Presentation elements and task rendering, as described in sections 4.3 and 4.4 respectively, are  
1677 used for notifications also. In most cases the subject line and description are sufficient information  
1678 for the recipients, especially if the notifications are received in an e-mail client or mobile device.  
1679 But in some cases the notifications can be received in a proprietary client so the notification may  
1680 support a proprietary rendering format to enable this to be utilized to the full, such as for  
1681 rendering data associated with the caller invoking the notification. For example, the description  
1682 could include a link to the process audit trail or a button to navigate to business transactions  
1683 involved in the underlying process.

1684 Notifications do not have ad-hoc attachments, comments or deadlines.

### 5.1 Overall Syntax

1685 Definition of notifications

```
1687 <htd:notification name="NCName">  
1688  
1689   <htd:interface portType="QName" operation="NCName"/>  
1690  
1691   <htd:priority expressionLanguage="anyURI"??>  
1692     integer-expression  
1693   </htd:priority>  
1694  
1695   <htd:peopleAssignments>  
1696  
1697     <htd:recipients>  
1698       ...  
1699     </htd:recipients>  
1700  
1701     <htd:businessAdministrators??>  
1702       ...  
1703     </htd:businessAdministrators>  
1704  
1705   </htd:peopleAssignments>  
1706  
1707   <htd:presentationElements>  
1708     ...  
1709   </htd:presentationElements>
```

```
1710
1711 <htd:renderings>?
1712   ...
1713 </htd:renderings>
1714
1715 </htd:notification>
```

## 1716 5.2 Properties

1717 The following attributes and elements are defined for notifications:

- 1718 • **name**: This attribute is used to specify the name of the notification. The name combined  
1719 with the target namespace of a notification element is used to uniquely identify the  
1720 notification definition. The attribute is mandatory. It is not used for notification rendering.
- 1721 • **interface**: This element is used to specify the operation used to invoke the notification.  
1722 The operation is specified using WSDL, that is a WSDL port type and WSDL operation  
1723 are defined. The element and its `portType` and `operation` attributes are mandatory.  
1724 The operation MUST be a one-way WSDL operation.
- 1725 • **priority**: This element is used to specify the priority of the notification. It is an optional  
1726 element which value is an integer expression. If not present, the priority of the task is  
1727 unspecified. 0 is the highest priority, larger numbers identify lower priorities. The result of  
1728 the expression evaluation is of type `xsd:integer`. The `expressionLanguage`  
1729 attribute specifies the language used in the expression. The attribute is optional. If not  
1730 specified, the default language as inherited from the closest enclosing element that  
1731 specifies the attribute is used.
- 1732 • **peopleAssignments**: This element is used to specify people assigned to the  
1733 notification. The element is mandatory. The element MUST include a people assignment  
1734 for recipients and MAY include a people assignment for business administrators.
- 1735 • **presentationElements**: The element is used to specify different information used to  
1736 display the notification, such as name, subject and description, in a task list. The element  
1737 is mandatory. See section 4.3 for more information on presentation elements.
- 1738 • **rendering**: The element is used to specify rendering method. It is optional. If not  
1739 present, notification rendering is implementation dependent. See section 4.4 for more  
1740 information on rendering.

## 1741 5.3 Notification Behavior and State Transitions

1742 Same as human tasks, notifications are in pseudo-state *Inactive* before they are activated. Once  
1743 they are activated they move to the *Ready* state. This state is observable, that is, when querying  
1744 for notifications then all notifications in state *Ready* are returned. When a notification is removed  
1745 then it moves into the final pseudo-state *Removed*.

1746

1747

## 6 Programming Interfaces

1748

### 6.1 Operations for Client Applications

1749

1750 A number of applications are involved in the life cycle of a task. These comprise:

- 1751 • The task list client, i.e. a client capable of displaying information about the task under
- 1752 consideration
- 1753 • The requesting application, i.e. any partner that has initiated the task
- 1754 • The supporting application, i.e. an application launched by the task list client to support
- 1755 processing of the task.

1756

1757 The task infrastructure provides access to a given task. It is important to understand that what is  
1758 meant by *task list client* is the software that presents a UI to one authenticated user, irrespective  
1759 of whether this UI is rendered by software running on server hardware (such as in a portals  
1760 environment) or client software (such as a client program running on a users workstation or PC).

1761 A given task exposes a set of operations to this end. A compliant implementation MUST provide  
1762 the operations listed below and an application (such as a task list client) may use these  
1763 operations to manipulate the task. All operations are executed in a synchronous fashion and  
1764 return faults provided that certain preconditions do not hold. The response message resulting  
1765 from an operation invocation may be void. The above applies to notifications also.

1766 An operation takes a well-defined set of parameters as its input. Passing an illegal parameter or  
1767 an illegal number of parameters results in the `illegalhta:illegalArgumentFault` being  
1768 thrown. Invoking an operation that is not allowed in the current state of the task results in an  
1769 `illegalhta:illegalStateFault`.

1770 By default, the identity of the person on behalf of which the operation is invoked is passed to the  
1771 task. When the person is not authorized to perform the operation the  
1772 `illegalhta:illegalAccessFault` and  
1773 `recipientNotAllowedhta:recipientNotAllowed` is thrown in the case of tasks and  
1774 notifications respectively.

1775 Invoking an operation that does not apply to the task type (e.g., invoking claim on a notification)  
1776 results in an `illegalhta:illegalOperationFault`.

1777 The language of the person on behalf of which the operation is invoked is assumed to be  
1778 available to operations requiring that information, e.g., when accessing presentation elements.

1779 \_\_\_\_\_ For an overview of which operations are allowed in what state, refer to section 4.7  
1780 "Human Task Behavior and State Transitions". For a formal definition of the allowed operations,  
1781 see [WS-HumanTask Data Types Schema](#)

1782 Note to specification editors: the WS-HumanTask data types XML Schema definition is separately  
1783 maintained in artifact

1784 [\\_ws-humantask-types.xsd](#)

1785 The contents of this artifact shall be copied back into this section before publishing the  
1786 specification, e.g., as a committee draft.

1787 [WS-HumanTask\\_API\\_Operations WSDL](#).

1788 This specification does not stipulate the authentication, language passing, addressing, and  
1789 binding scheme employed when calling an operation. This can be achieved using different  
1790 mechanisms (e.g. WS-Security, WS-Addressing).

Formatted: Normal

Formatted: Normal

Formatted: Font: Arial

Formatted: Font: Courier New

1791 **6.1.1 Participant Operations**

1792 Operations are executed by end users, i.e. actual or potential owners. The identity of the user is  
 1793 implicitly passed when invoking any of the operations listed in the table below. The participant  
 1794 operations listed below only apply to tasks unless explicitly noted otherwise. The authorization  
 1795 column indicates people of which roles are authorized to perform the operation. Stakeholders of  
 1796 the task are not mentioned explicitly. They have the same authorization rights as business  
 1797 administrators.  
 1798

Operation Name	Description	Parameters	Authorization
claim	Claim responsibility for a task, i.e. set the task to status <i>Reserved</i>	In <ul style="list-style-type: none"> <li>• task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>• void</li> </ul>	Potential Owners Business Administrator
start	Start the execution of the task, i.e. set the task to status <i>InProgress</i> .	In <ul style="list-style-type: none"> <li>• task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>• void</li> </ul>	Actual Owner Potential Owners (state <i>Ready</i> )
stop	Cancel/stop the processing of the task. The task returns to the <i>Reserved</i> state.	In <ul style="list-style-type: none"> <li>• task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>• void</li> </ul>	Actual Owner Business Administrator
release	Release the task, i.e. set the task back to status <i>Ready</i> .	In <ul style="list-style-type: none"> <li>• task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>• void</li> </ul>	Actual Owner Business Administrator
suspend	Suspend the task.	In <ul style="list-style-type: none"> <li>• task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>• void</li> </ul>	Potential Owners (state <i>Ready</i> ) 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 <ul style="list-style-type: none"> <li>• task identifier</li> <li>• time period</li> <li>• point of time</li> </ul> Out <ul style="list-style-type: none"> <li>• void</li> </ul>	Potential Owners (state <i>Ready</i> ) Actual Owner Business Administrator
resume	Resume a suspended task.	In <ul style="list-style-type: none"> <li>• task identifier</li> </ul>	Potential Owners (state <i>Ready</i> ) Actual Owner

		Out <ul style="list-style-type: none"> <li>void</li> </ul>	Business Administrator
complete	Execution of the task finished successfully. If no output data is set the operation returns <a href="#">illegalhta:illegalArgumentFault</a> .	In <ul style="list-style-type: none"> <li>task identifier</li> <li>output data of task</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>	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 <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>	Notification Recipient
fail	Actual owner completes the execution of the task raising a fault. The fault <a href="#">illegalhta:illegalOperationFault</a> is returned if the task interface defines no faults. If fault name or fault data is not set the operation returns <a href="#">illegalhta:illegalArgumentFault</a> .	In <ul style="list-style-type: none"> <li>task identifier</li> <li>fault name</li> <li>fault data</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>	Actual Owner
setPriority	Change the priority of the task. The caller has to specify the integer value of the new priority.	In <ul style="list-style-type: none"> <li>task identifier</li> <li>priority</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>	Actual Owner Business Administrator
addAttachment	Add attachment to a task.	In <ul style="list-style-type: none"> <li>task identifier</li> <li>attachment name</li> <li>access type</li> <li>attachment</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>	Actual Owner Business Administrator

getAttachmentInfos	Get attachment information for all attachments associated with the task.	In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>list of attachment data (list of <del>htttta</del>:attachmentInfo)</li> </ul>	Potential Owners Actual Owner Business Administrator
getAttachments	Get all attachments of a task with a given name.	In <ul style="list-style-type: none"> <li>task identifier</li> <li>attachment name</li> </ul> Out <ul style="list-style-type: none"> <li>list of attachments (list of <del>htttta</del>htt:attachment)</li> </ul>	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 <ul style="list-style-type: none"> <li>task identifier</li> <li>attachment name</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>	Actual Owner Business Administrator
addComment	Add a comment to a task.	In <ul style="list-style-type: none"> <li>task identifier</li> <li>plain text</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>	Potential Owners Actual Owner Business Administrator
getComments	Get all comments of a task	In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>list of comments (list of <del>htttta</del>htt:comment)</li> </ul>	Potential Owners Actual Owner Business Administrator
skip	Skip the task. If the task is not skipable then the fault <del>illegalhta:illegal</del> OperationFault is returned.	In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out <ul style="list-style-type: none"> <li>void</li> </ul>	Task Initiator Actual Owner Business Administrator
forward	Forward the task to	In	Potential Owners

	<p>another organization entity. The caller has to specify the receiving organizational entity.</p> <p>Potential owners can only forward a task while the task is in the <i>Ready</i> state.</p> <p>For details on forwarding human tasks refer to section 4.7.3.</p>	<p>Out</p> <ul style="list-style-type: none"> <li>task identifier</li> <li>organizational entity (<code>htd:tOrganizationalEntity</code>)</li> <li>void</li> </ul>	Actual Owner Business Administrator
delegate	<p>Assign the task to one user and set the task to state <i>Reserved</i>. If the recipient was not a potential owner then this person is added to the set of potential owners.</p> <p>For details on delegating human tasks refer to section 4.7.3.</p>	<p>In</p> <ul style="list-style-type: none"> <li>task identifier</li> <li>organizational entity (<code>htd:tOrganizationalEntity</code>)</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>void</li> </ul>	Potential Owners (only in <i>Ready</i> state) Actual Owner Business Administrator
getRendering	<p>Applies to both tasks and notifications.</p> <p>Returns the rendering specified by the type parameter.</p>	<p>In</p> <ul style="list-style-type: none"> <li>task identifier</li> <li>rendering type</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>any type</li> </ul>	Any
getRenderingTypes	<p>Applies to both tasks and notifications.</p> <p>Returns the rendering types available for the task or notification.</p>	<p>In</p> <ul style="list-style-type: none"> <li>task identifier</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>list of QNames</li> </ul>	Any
getTaskInfo	<p>Applies to both tasks and notifications.</p> <p>Returns a data object of type <code>htt:tTask</code></p>	<p>In</p> <ul style="list-style-type: none"> <li>task identifier</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>task (<del>htd</del><code>htt:tTask</code>)</li> </ul>	Any
getTaskDescription	<p>Applies to both tasks and notifications. Returns the presentation description in the specified mime type.</p>	<p>In</p> <ul style="list-style-type: none"> <li>task identifier</li> <li>content type – optional, default is text/plain</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>string</li> </ul>	Any
setOutput	<p>Set the data for the part of the task's output message.</p>	<p>In</p> <ul style="list-style-type: none"> <li>task identifier</li> <li>part name (optional for</li> </ul>	Actual Owner

		<ul style="list-style-type: none"> <li>single part messages )</li> <li>• output data of task</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>• void</li> </ul>	
deleteOutput	Deletes the output data of the task.	<p>In</p> <ul style="list-style-type: none"> <li>• task identifier</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>• void</li> </ul>	Actual Owner
setFault	Set the fault data of the task. The fault <u>illegalhta:illegalOperationFault</u> is returned if the task interface defines no faults.	<p>In</p> <ul style="list-style-type: none"> <li>• task identifier</li> <li>• fault name</li> <li>• fault data of task</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>• void</li> </ul>	Actual Owner
deleteFault	Deletes the fault name and fault data of the task.	<p>In</p> <ul style="list-style-type: none"> <li>• task identifier</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>• void</li> </ul>	Actual Owner
getInput	Get the data for the part of the task's input message.	<p>In</p> <ul style="list-style-type: none"> <li>• task identifier</li> <li>• part name (optional for single part messages)</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>• any type</li> </ul>	Potential Owners Actual owner Business Administrator
getOutput	Get the data for the part of the task's output message.	<p>In</p> <ul style="list-style-type: none"> <li>• task identifier</li> <li>• part name (optional for single part messages)</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>• any type</li> </ul>	Actual Owner Business Administrator
getFault	Get the fault data of the task.	<p>In</p> <ul style="list-style-type: none"> <li>• task identifier</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>• fault name</li> <li>• fault data</li> </ul>	Actual Owner Business Administrator
<a href="#">getOutcome</a>	<a href="#">Get the outcome of the task</a>	<p><a href="#">In</a></p> <ul style="list-style-type: none"> <li>• <a href="#">task identifier</a></li> </ul> <p><a href="#">Out</a></p>	<a href="#">Any</a>

Formatted: Bullets and Numbering

1799 All these operations MUST be supported by a compliant implementation.

1800 **6.1.2 Simple Query Operations**

1801 Simple query operations allow to retrieve task data. These operations MUST be supported by a  
 1802 compliant implementation. The identity of the user is implicitly passed when invoking any of the  
 1803 following operations.

1804

Operation Name	Description	Parameters	Authorization
getMyTaskAbstracts	<p>Retrieve the task abstracts. This operation is used to obtain the data required to display a task list.</p> <p>If no 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.</p> <p>The <i>where</i> clause may only reference exactly one column using the following operators: <i>equals</i> (“=”), <i>not equals</i> (“&lt;&gt;”), <i>less than</i> (“&lt;”), <i>greater than</i> (“&gt;”), <i>less than or equals</i> (“&lt;=”), and <i>greater than or equals</i> (“&gt;=”), e.g., “Task.Priority = 1”.</p> <p>The <i>where</i> clause is logically ANDed with the created-on clause, which may only reference the column Task.CreatedOn with operators as described above.</p> <p>The combination of the two clauses enables simple but restricted paging in a task list client.</p> <p>If maxTasks is specified, then the number of task abstracts returned for this query will not exceed this limit.</p>	<p>In</p> <ul style="list-style-type: none"> <li>task type (“ALL”   “TASKS”   “NOTIFICATIONS”)</li> <li>generic human role</li> <li>work queue</li> <li>status list</li> <li>where clause</li> <li>created-on clause</li> <li>maxTasks</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>list of tasks (list of <del>http</del>:tTaskAbstract)</li> </ul>	Any

getMyTasks	<p>Retrieve the task details. This operation is used to obtain the data required to display a task list, as well as the details for the individual tasks.</p> <p>If no 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.</p> <p>The <i>where</i> clause may only reference exactly one column using the following operators: <i>equals</i> (“=”), <i>not equals</i> (“&lt;&gt;”), <i>less than</i> (“&lt;”), <i>greater than</i> (“&gt;”), <i>less than or equals</i> (“&lt;=”), and <i>greater than or equals</i> (“&gt;=”), e.g., “Task.Priority = 1”.</p> <p>The <i>where</i> clause is logically ANDed with the created-on clause, which may only reference the column Task.CreatedOn with operators as described above.</p> <p>The combination of the two clauses enables simple but restricted paging in the task list client.</p> <p>If maxTasks is specified, then the number of task details returned for this query will not exceed this limit.</p>	<p>In</p> <ul style="list-style-type: none"> <li>• task type (“ALL”   “TASKS”   “NOTIFICATIONS”)</li> <li>• generic human role</li> <li>• work queue</li> <li>• status list</li> <li>• where clause</li> <li>• created-on clause</li> <li>• maxTasks</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>• list of tasks (list of <del>tt</del> <code>tt: tTask</code>)</li> </ul>	Any
------------	--	---	-----

1805

1806 The return types tTaskAbstract and tTask are defined in section 3.4.4 “Data Types for Task  
1807 Instance Data”.

1808

1809 **Simple Task View**

1810 The table below lists the task attributes available to the simple query operations. This view is  
1811 used when defining the where clause of any of the above query operations.

1812

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

1813

### 1814 6.1.3 Advanced Query Operation

1815 The advanced query operation is used by the task list client to perform queries not covered by the  
 1816 simple query operations defined in 6.1.2. A compliant implementation MAY support this operation.  
 1817 An implementation MAY restrict the results according to authorization of the invoking user.

1818

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 <ul style="list-style-type: none"> <li>• select clause</li> <li>• where clause</li> <li>• order-by clause</li> <li>• maxTasks</li> </ul>

	<p>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.</p>	<p>Out</p> <ul style="list-style-type: none"> <li>taskIndexOffset</li> <li>query result (<a href="#">htd:taskQueryResultSet</a>/<a href="#">tTaskQueryResultSet</a>)</li> </ul>
--	--	---

1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831

**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"
```

```

1862         type="tPresentationSubject"/>
1863     <xsd:element name="renderingMethodExists" type="xsd:boolean"/>
1864     <xsd:element name="hasOutput" type="xsd:boolean"/>
1865     <xsd:element name="hasFault" type="xsd:boolean"/>
1866     <xsd:element name="hasAttachments" type="xsd:boolean"/>
1867     <xsd:element name="hasComments" type="xsd:boolean"/>
1868     <xsd:element name="escalated" type="xsd:boolean"/>
1869     <xsd:element name="primarySearchBy" type="xsd:string"/>
1870     <xsd:element name="outcome" type="xsd:string"/>
1871     <xsd:any namespace="##other" processContents="lax"/>
1872 </xsd:choice>
1873 </xsd:complexType>

```

1874

### 1875 Complete Task View

1876 The table below is the set of columns used when defining select clause, where clause, and order-  
1877 by clause of query operations. Conceptually, this set of columns defines a universal relation. As a  
1878 result the query can be formulated without specifying a from clause. A compliant implementation  
1879 MAY extend this view by adding columns.

1880

Column Name	Type	Constraints
ID	xsd:string	
TaskType	Enumeration	Identifies the task type. The following values are allowed: <ul style="list-style-type: none"> <li>“TASK” for a human task</li> <li>“NOTIFICATION” for notifications</li> </ul> Note that notifications are simple tasks that do not block the progress of the caller,
Name	xsd:Qname	
Status	Enumeration	For values see section 4.7 “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.
Pres <del>entation</del> Name	xsd:string	The task's presentation name.
Pres <del>entation</del> Subject	xsd:string	The task's presentation subject.
RenderingMethod <del>od</del> Name	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	
<a href="#">Outcome</a>	<a href="#">xsd:string</a>	

1881

#### 1882 6.1.4 Administrative Operations

1883 Operations to be executed for administrative purposes. Actual definition of authorization for  
 1884 operations is outside the scope of this specification.

1885

Operation Name	Description	Parameters	Authorization
activate	Activate the task, i.e. set the task to status <i>Ready</i> .	In <ul style="list-style-type: none"> <li>task identifier</li> </ul> Out	Business Administrator

		<ul style="list-style-type: none"> <li>void</li> </ul>	
nominate	Nominate an organization entity to process the task. If it is nominated to one person then the new state of the task is <i>Reserved</i> . If it is nominated to several people then the new state of the task is <i>Ready</i> . This can only be performed when the task is in the state <i>Created</i> .	<p>In</p> <ul style="list-style-type: none"> <li>task identifier</li> <li>organizational entity (htd:<del>OrganizationalType</del>OrganizationalEntity)</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>void</li> </ul>	Business Administrator
setGenericHumanRole	Replace the organizational assignment to the task in one generic human role.	<p>In</p> <ul style="list-style-type: none"> <li>task identifier</li> <li>generic human role</li> <li>organizational entity (htd:<del>OrganizationalType</del>OrganizationalEntity)</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>void</li> </ul>	Business Administrator

1886

## 1887 6.2 XPath Extension Functions

1888 [This section introduces](#)The following XPath extension functions [that](#) are provided to be used  
1889 within the definition of a human task or notification. When defining properties using these XPath  
1890 functions note the initialization order in section 4.7.1.

1891 [Definition of these XPath extension functions is provided in the table below. Input parameters that](#)  
1892 [specify task name, message part name or logicalPeopleGroup name MUST be literal strings. This](#)  
1893 [restriction does not apply to other parameters.](#) Because XPath 1.0 functions do not support  
1894 returning faults, an empty node set is returned in the event of an error.

1895 XPath functions used for notifications in an escalation can access context from the enclosing task  
1896 by specifying that task's name.

1897

Operation Name	Description	Parameters
getPotentialOwners	Returns the potential owners of the task. Evaluates to an empty htd:organizationalEntity in case of an error.	<p>In</p> <ul style="list-style-type: none"> <li>task name (optional)</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>potential owners</li> </ul>

	If the task name is not present the current task is considered.	( <code>htd:organizationalEntity</code> )
<code>getActualOwner</code>	Returns the actual owner of the task. Evaluates to an empty <code>htd:user</code> in case there is no actual owner. If the task name is not present the current task is considered.	In <ul style="list-style-type: none"> <li>task name (optional)</li> </ul> Out <ul style="list-style-type: none"> <li>the actual owner (user id as <code>htd:user</code>)</li> </ul>
<code>getTaskInitiator</code>	Returns the initiator of the task. Evaluates to an empty <code>htd:user</code> in case there is no initiator. If the task name is not present the current task is considered.	In <ul style="list-style-type: none"> <li>task name (optional)</li> </ul> Out <ul style="list-style-type: none"> <li>the task initiator (user id as <code>htd:user</code>)</li> </ul>
<code>getTaskStakeholders</code>	Returns the stakeholders of the task. Evaluates to an empty <code>htd:organizationalEntity</code> in case of an error. If the task name is not present the current task is considered.	In <ul style="list-style-type: none"> <li>task name (optional)</li> </ul> Out <ul style="list-style-type: none"> <li>task stakeholders (<code>htd:organizationalEntity</code>)</li> </ul>
<code>getBusinessAdministrators</code>	Returns the business administrators of the task. Evaluates to an empty <code>htd:organizationalEntity</code> in case of an error. If the task name is not present the current task is considered.	In <ul style="list-style-type: none"> <li>task name (optional)</li> </ul> Out <ul style="list-style-type: none"> <li>business administrators (<code>htd:organizationalEntity</code>)</li> </ul>
<code>getExcludedOwners</code>	Returns the excluded owners. Evaluates to an empty <code>htd:organizationalEntity</code> in case of an error. If the task name is not present the current task is considered.	In <ul style="list-style-type: none"> <li>task name (optional)</li> </ul> Out <ul style="list-style-type: none"> <li>excluded owners (<code>htd:organizationalEntity</code>)</li> </ul>
<code>getTaskPriority</code>	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 <ul style="list-style-type: none"> <li>task name (optional)</li> </ul> Out <ul style="list-style-type: none"> <li>priority (<a href="#">xsd:nonNegativeInteger</a>)</li> </ul>

getInput	Returns the part of the task's input message. If the task name is not present the current task is considered.	In <ul style="list-style-type: none"> <li>part name</li> <li>task name (optional)</li> </ul> Out <ul style="list-style-type: none"> <li>input message</li> </ul>
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 <code>htd:organizationalEntity</code> contains an empty user list. If the task name is not present the current task is considered.	In <ul style="list-style-type: none"> <li>task name (optional)</li> <li>name of the logical people group</li> </ul> Out <ul style="list-style-type: none"> <li>the value of the logical people group (<code>htd:organizationalEntity</code>)</li> </ul>
<a href="#">getOutcome</a>	<a href="#">Returns the outcome of the task. Evaluates to an empty string in case there is no outcome specified for the task.</a> <a href="#">If the task name is not present the current task is considered.</a>	In <ul style="list-style-type: none"> <li><a href="#">task name (optional)</a></li> </ul> Out <ul style="list-style-type: none"> <li><a href="#">the task outcome</a> (<code>xsd:string</code>)</li> </ul>
union	Constructs an <code>organizationalEntity</code> containing every user that occurs in <b>either set1 or set2</b> , eliminating duplicate users.	In <ul style="list-style-type: none"> <li>set1 (<code>htd:organizationalEntity</code> <code> htd:users</code> <code> htd:user</code>)</li> <li>set2 (<code>htd:organizationalEntity</code> <code> htd:users</code> <code> htd:user</code>)</li> </ul> Out <ul style="list-style-type: none"> <li>result (<code>htd:organizationalEntity</code>)</li> </ul>
intersect	Constructs an <code>organizationalEntity</code> containing every user that occurs in <b>both set1 and set2</b> , eliminating duplicate users.	In <ul style="list-style-type: none"> <li>set1 (<code>htd:organizationalEntity</code> <code> htd:users</code> <code> htd:user</code>)</li> <li>set2 (<code>htd:organizationalEntity</code> <code> htd:users</code> <code> htd:user</code>)</li> </ul>

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Font: Courier New

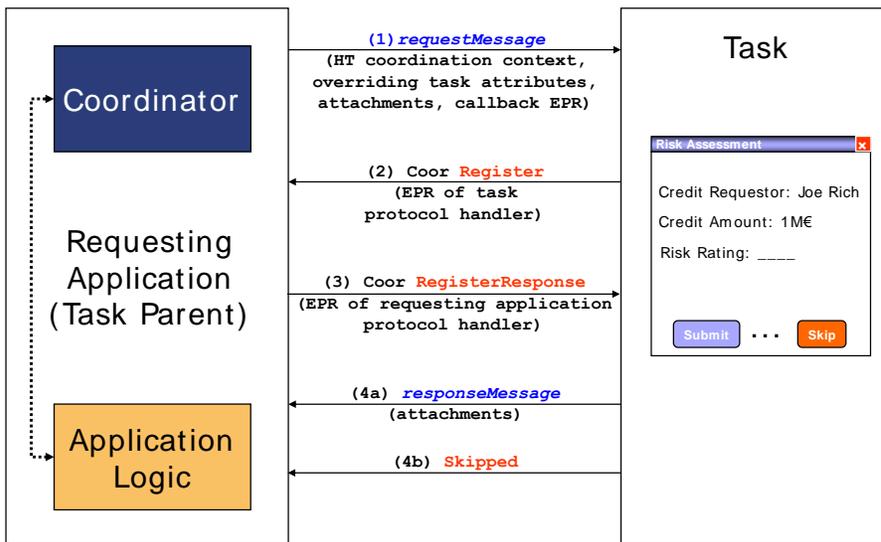
		<p>ity  htd:users  htd:user)</p> <p>Out</p> <ul style="list-style-type: none"> <li>• <b>result</b> (htd:organizationalEnt ity)</li> </ul>
Except	<p>Constructs an organizationalEntity containing every user that occurs in <b>set1 but not in set2</b>.</p> <p>Note: This function is required to allow enforcing the separation of duties ("4-eyes principle").</p>	<p>In</p> <ul style="list-style-type: none"> <li>• <b>set1</b> (htd:organizationalEnt ity  htd:users  htd:user)</li> <li>• <b>set2</b> (htd:organizationalEnt ity  htd:users  htd:user)</li> </ul> <p>Out</p> <ul style="list-style-type: none"> <li>• <b>result</b> (htd:organizationalEnt ity)</li> </ul>

1898

1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907

## 7 Interoperable Protocol for Advanced Interaction with Human Tasks

Previous sections describe how to define standard invocable Web services that happen to be implemented by human tasks or notifications. Additional capability results from an application that is human task aware, and can control the autonomy and life cycle of the human tasks. To address this in an interoperable manner, a coordination protocol, namely the *WS-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.



1908

Figure 1: Message Exchange between Application and Human Task

1909 While we do not make any assumptions about the nature of the application in the following  
1910 scenarios, in practice it would be hosted by an infrastructure that actually deals with the WS-HT  
1911 coordination protocol on the application's behalf.

1912 In case of human tasks the following message exchanges are possible.

1913 **Scenario 1:** At some point in time, the application invokes the human task through its service  
1914 interface. In order to signal to the human task infrastructure that an instance of the human task  
1915 should be created which is actually coordinated by the parent application, this request message  
1916 contains certain control information. This control information consists of a coordination context of  
1917 the WS-HT coordination protocol, and optional human task attributes that are used to override  
1918 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->

1922 [http://docs.oasis-open.org/ns/bpel4people/ws-  
1924 humantask/protocol/200803](http://docs.oasis-open.org/ns/bpel4people/ws-<br/>1923 humantask/protocol/200803). The inclusion of a coordination context within the  
1925 request message indicates that the life cycle of the human tasks is managed via  
1926 corresponding protocol messages from outside its hosting WS-HumanTask (WS-HT)  
1927 implementation. The coordination context further contains in its `RegistrationService`  
1928 element an endpoint reference that the WS-HT implementation hosting the human task  
1929 must use to register the task as participant of that coordination type.  
1930 Note: In a typical implementation, the parent application or its environment will create that  
1931 coordination context by issuing an appropriate request against the WS-Coordination (WS-  
1932 C) activation service, followed by registering the parent application as a `TaskParent`  
participant in that protocol.

- 1933 • The optional human task attributes allow overriding aspects of the definition of the human  
1934 task from the calling application. The calling application may set values of the following  
1935 attributes of the task definition:
  - 1936 ○ Priority of the task
  - 1937 ○ Actual people assignments for each of the generic human roles of the human  
1938 task
  - 1939 ○ The skipable indicator which determines whether a task can actually be skipped  
1940 at runtime.
  - 1941 ○ The amount of time by which the task activation is deferred.
  - 1942 ○ The expiration time for the human task after which the calling application is no  
1943 longer interested in its result.

← Formatted: Bullets and Numbering

1944 After having created this request message, it is sent to the WS-HT implementation hosting the  
1945 human task (step (1) in Figure 1). The WS-HT implementation receiving that message extracts  
1946 the coordination context and callback information, the human task attributes (if present) and the  
1947 application payload. Before passing this application payload to the human task, the WS-HT  
1948 implementation registers the human task to be created with the registration service passed as  
1949 part of the coordination context (step (2) in Figure 1). The corresponding WS-C `Register`  
1950 message includes the endpoint reference (EPR) of the protocol handler of the WS-HT  
1951 implementation of the human task that the parent application must use to send all protocol  
1952 messages to. This EPR is the value contained in the `ParticipantProtocolService` element  
1953 of the `Register` message. Furthermore, the registration MUST be as `HumanTask` participant by  
1954 specifying the corresponding value in the `ProtocolIdentifier` element of the `Register`  
1955 message. The parent application reacts to that message by sending back a `RegisterResponse`  
1956 message. This message contains in its `CoordinatorProtocolService` element the EPR of  
1957 the protocol handler of the parent application, which is used by the WS-HT implementation of the  
1958 human task for sending protocol messages to the parent application (step (3) in Figure 1).

1959 Now the instance of the human task is activated, so the assigned person can perform the task  
1960 (e.g. the risk assessment). Once the human task was successfully completed, a response  
1961 message is passed back to the parent application (step (4a) in Figure 1).

1962

1963 **Scenario 2:** If the human task is not completed with a result, but the assigned person determines  
1964 that the task should rather be skipped (and hence reaches its *Obsolete* final state), a “skipped”  
1965 coordination protocol message is sent from the human task to its parent application (step (4b) in  
1966 Figure 1). No response message is passed back.

1967

1968 **Scenario 3:** If the parent application needs to end prematurely before the invoked human task  
1969 has been completed, it sends an `exit` coordination protocol message to the human task, causing  
1970 the human task to end its processing. No response message is passed back.

1971

1972 In case of notifications, only some of the overriding attributes are propagated with the request  
 1973 message. Only priority and people assignments can be overridden for a notification, and the  
 1974 elements isSkipable, expirationTime and attachments are ignored if present. Likewise, the WS-  
 1975 HT coordination context, attachments and the callback EPR do not apply to notifications and are  
 1976 ignored as well. Finally, a notification does not return WS-HT coordination protocol messages.  
 1977 There is no message exchange beyond the initiating request message.

## 1978 7.1 Human Task Coordination Protocol Messages

1979 The following section describes the behavior of the human task with respect to the protocol  
 1980 messages exchanged with its requesting application which is human task aware. In particular, we  
 1981 describe which state transitions trigger which protocol message and vice versa. Human task  
 1982 aware requesting applications MUST support WS-HT protocol messages in addition to application  
 1983 requesting, responding and fault messages.

1984 See diagram in section 4.7 “Human Task Behavior and State Transitions”.

- 1985 1. The initiating message containing a WS-HT coordination context is received by the  
 1986 hosting WS-HT implementation. This message may also include ad hoc attachments that  
 1987 are to be made available to the task processor. A new task is created. As part of the  
 1988 context, an EPR of the registration service is passed. This registration service MUST be  
 1989 used by the hosting WS-HT implementation to register the protocol handler receiving the  
 1990 WS-HT protocol messages sent by the requesting application’s implementation. If an  
 1991 error occurs during the task instantiation the final state *Error* is reached and protocol  
 1992 message *fault* is sent to the requesting application.
- 1993 2. On successful completion of the task an application level response message is sent and  
 1994 the task moves to state *Completed*. When this happens, attachments created during the  
 1995 processing of the task may be added to the response message. Attachments that had  
 1996 been passed in the initiating message are not returned.
- 1997 3. On unsuccessful completion (completion with a fault message), an application level fault  
 1998 message is sent and the task moves to state *Failed*. When this happens, attachments  
 1999 created during the processing of the task are added to the response message.  
 2000 Attachments that had been passed in the initiating message are not returned.
- 2001 4. If the task experiences a non-recoverable error protocol message *fault* is sent and  
 2002 the task moves to state *Error*. No attachments are returned.
- 2003 5. If the task is skipable and is skipped then the task sends the protocol message  
 2004 *skipped* and it moves to state *Obsolete*. No attachments are returned.
- 2005 6. On receipt of protocol message *exit* the task exits. This indicates that the requesting  
 2006 application is no longer interested in any result produced by the task. No attachments are  
 2007 returned.

2008 The following table summarizes this behavior, the messages sent, and their direction, i.e.,  
 2009 whether a message is sent from the requesting application to the task (“out” in the column titled  
 2010 Direction) or vice versa (“in”).

2011

Message	Direction	Human Task Behavior ( and Protocol messages)
application request with WS-HT coordination context	in	Create task <sub>7</sub> (Register)
application response	out	Successful completion with response
application fault response	out	Completion with fault response
<a href="#">htcp:Fault</a>	out	Non-recoverable error

<code>htcp:Exit</code>	in	Requesting application is no longer interested in the task output
<code>htcp:Skipped</code>	out	Task moves to state <i>Obsolete</i>

## 2012 7.2 Protocol Messages

2013 All WS-HT protocol messages have the following type:

```
2014 <xsd:complexType name="ProtocolMsgType">
2015   <xsd:sequence>
2016     <xsd:any namespace="##other" processContents="lax"
2017       minOccurs="0" maxOccurs="unbounded" />
2018   </xsd:sequence>
2019   <xsd:anyAttribute namespace="##other" processContents="lax" />
2020 </xsd:complexType>
```

2022 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.

### 2024 7.2.1 Protocol Messages Received by a Task Parent

2025 The following is the definition of the `htcp:skipped` message.

```
2026 <xsd:element name="skipped"
2027   type="spec:ProtocolMsgTypehtcp:ProtocolMsgType" />
2028 <wsdl:message name="skipped">
2029   <wsdl:part name="parameters" element="htcp:skipped" />
2030 </wsdl:message>
```

2031 The `htcp:skipped` message is used to inform the task parent (i.e. the requesting application) that the invoked task has been skipped. The task does not return any result.

2033 The following is the definition of the `htcp:fault` message.

```
2034 <xsd:element name="fault" type="spec:ProtocolMsgTypehtcp:ProtocolMsgType"
2035   />
2036 <wsdl:message name="fault">
2037   <wsdl:part name="parameters" element="htcp:fault" />
2038 </wsdl:message>
```

2040 The `htcp:fault` message is used to inform the task parent that the task has ended abnormally. The task does not return any result.

### 2042 7.2.2 Protocol Messages Received by a Task

2043 Upon receipt of the following `htcp:exit` message the task parent informs the task that it is no longer interested in its results.

```
2045 <xsd:element name="exit" type="spec:ProtocolMsgTypehtcp:ProtocolMsgType"
2046   />
2047 <wsdl:message name="exit">
2048   <wsdl:part name="parameters" element="htcp:exit" />
2049 </wsdl:message>
```

## 2050 7.3 WSDL of the Protocol Endpoints

2051 Protocol messages are received by protocol participants via operations of dedicated ports called  
2052 protocol endpoints. In this section we specify the WSDL port types of the protocol endpoints  
2053 needed to run the WS-HT coordination protocol.

### 2054 7.3.1 Protocol Endpoint of the Task Parent

2055 An application that wants to create a task and wants to become a task parent must provide an  
2056 endpoint implementing the following port type. This endpoint is the protocol endpoint of the task  
2057 parent receiving protocol messages of the WS-HT coordination protocol from a task. The  
2058 operation used by the task to send a certain protocol message to the task parent is named by the  
2059 message name of the protocol message concatenated by the string *Operation*. For example,  
2060 the *skipped* message can be passed to the tasked parent by using the operation named  
2061 *skippedOperation*.

```
2062 <wsdl:portType name="clientParticipantPortType">
2063   <wsdl:operation name="skippedOperation">
2064     <wsdl:input message="htdphtcp:skipped" />
2065   </wsdl:operation>
2066   <wsdl:operation name="faultOperation">
2067     <wsdl:input message="htdphtcp:fault" />
2068   </wsdl:operation>
2069 </wsdl:portType>
```

### 2070 7.3.2 Protocol Endpoint of the Task

2071 A task must provide an endpoint implementing the following port type. This endpoint is the  
2072 protocol endpoint of the task receiving protocol messages of the WS-HT coordination protocol  
2073 from a task parent. The operation used by the task parent to send a certain protocol message to  
2074 a task is named by the message name of the protocol message concatenated by the string  
2075 *Operation*. For example, the *exit* protocol message can be passed to the subprocess by  
2076 using the operation named *exitOperation*.

```
2077 <wsdl:portType name="humanTaskParticipantPortType">
2078   <wsdl:operation name="exitOperation">
2079     <wsdl:input message="htdphtcp:exit" />
2080   </wsdl:operation>
2081 </wsdl:portType>
```

## 2082 7.4 Providing Human Task Context

2083 The task context information is exchanged between the requesting application and a task or a  
2084 notification. In case of tasks, this information is passed as header fields of the request and  
2085 response messages of the task's operation. In case of notifications, this information is passed as  
2086 header fields of the request message of the notification's operation.

### 2087 7.4.1 Schema of the Human Task Context

2088 The following describes the XML schema representation of the task context:

```
2089 <xsd:element name="humanTaskContext" type="tHumanTaskContext" />
2090 <xsd:complexType name="tHumanTaskContext">
2091   <xsd:sequence>
2092     <xsd:element name="priority" type="xsd:nonNegativeInteger"
2093       minOccurs="0" />
2094     <xsd:element name="peopleAssignments" type="tPeopleAssignments"
2095       minOccurs="0" />
2096   </xsd:sequence>
2097 </xsd:complexType>
```

```

2096 <xsd:element name="isSkipable" type="xsd:boolean" minOccurs="0" />
2097 <xsd:element name="expirationTime" type="xsd:dateTime"
2098     minOccurs="0" />
2099 <xsd:element name="attachments" type="tAttachments" minOccurs="0" />
2100 </xsd:sequence>
2101 </xsd:complexType>
2102
2103 <xsd:complexType name="tPeopleAssignments">
2104 <xsd:sequence>
2105 <xsd:group ref="genericHumanRole" minOccurs="0"
2106     maxOccurs="unbounded" />
2107 </xsd:sequence>
2108 </xsd:complexType>
2109
2110 <xsd:group name="genericHumanRole">
2111 <xsd:choice>
2112 <xsd:element ref="potentialOwners" />
2113 <xsd:element ref="excludedOwners" />
2114 <xsd:element ref="taskInitiator" />
2115 <xsd:element ref="taskStakeholders" />
2116 <xsd:element ref="businessAdministrators" />
2117 <xsd:element ref="recipients" />
2118 </xsd:choice>
2119 </xsd:group>
2120 <xsd:element name="potentialOwners" type="tGenericHumanRole" />
2121 <xsd:element name="excludedOwners" type="tGenericHumanRole" />
2122 <xsd:element name="taskInitiator" type="tGenericHumanRole" />
2123 <xsd:element name="taskStakeholders" type="tGenericHumanRole" />
2124 <xsd:element name="businessAdministrators" type="tGenericHumanRole" />
2125 <xsd:element name="recipients" type="tGenericHumanRole" />
2126 <xsd:complexType name="tGenericHumanRole">
2127 <xsd:sequence>
2128 <xsd:element ref="htd:organizationalEntity" />
2129 </xsd:sequence>
2130 </xsd:complexType>
2131
2132 <xsd:complexType name="tAttachments">
2133 <xsd:sequence>
2134 <xsd:element name="returnAttachments" type="tReturnAttachments"
2135     minOccurs="0" />
2136 <xsd:element ref="hta:attachment" minOccurs="0"
2137     maxOccurs="unbounded" />
2138 </xsd:sequence>
2139 </xsd:complexType>
2140
2141 <xsd:simpleType name="tReturnAttachments">
2142 <xsd:restriction base="xsd:string">
2143 <xsd:enumeration value="all" />
2144 <xsd:enumeration value="newOnly" />
2145 <xsd:enumeration value="none" />
2146 </xsd:restriction>
2147 </xsd:simpleType>

```

Formatted: English (United States)

## 2148 7.4.2 SOAP Binding of Human Task Context

2149 In general, a SOAP binding specifies for message header fields how they are bound to SOAP  
2150 headers. In case of WS-HumanTask, the humanTaskContext element is simply mapped to a

2151 single SOAP header as a whole. The following listing shows the SOAP binding of the human task  
2152 context in an infoset representation.

```
2153 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"  
2154           xmlns:htdphtc="http://www.osoa.org/WS-  
2155 HT/protocolhttp://docs.oasis-open.org/ns/bpel4people/ws-  
2156 humantask/context/200803">  
2157   <S:Header>  
2158     <htdphtc:humanTaskContext>  
2159       <htdphtc:priority>...</htdphtc:priority>?  
2160       <htdphtc:peopleAssignments>...</htdphtc:peopleAssignments>?  
2161       <htdphtc:isSkipable>...</htdphtc:isSkipable>?  
2162       <htdphtc:expirationTime>...</htdphtc:expirationTime>?  
2163       <htdphtc:attachments>...</htdphtc:attachments>?  
2164     </htdphtc:humanTaskContext>  
2165   </S:Header>  
2166   <S:Body>  
2167     ...  
2168   </S:Body>  
2169 </S:Envelope>
```

- Formatted: French (France)
- Formatted: French (Canada)
- Formatted: English (United States)

2171 The following listing is an example of a SOAP message containing a human task context.

```
2172 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"  
2173           xmlns:htdphtc="http://www.osoa.org/WS-  
2174 HT/protocolhttp://docs.oasis-open.org/ns/bpel4people/ws-  
2175 humantask/context/200803">  
2176   <S:Header>  
2177     <htdphtc:humanTaskContext>  
2178       <htdphtc:priority>0</htdphtc:priority>  
2179       <htdphtc:peopleAssignments>  
2180         <htdphtd:potentialOwners>  
2181           <htdphtd:organizationalEntity>  
2182             <htdphtd:users>  
2183               <htdphtd:user>Alan</htdphtd:user>  
2184               <htdphtd:user>Dieter</htdphtd:user>  
2185               <htdphtd:user>Frank</htdphtd:user>  
2186               <htdphtd:user>Gerhard</htdphtd:user>  
2187               <htdphtd:user>Ivana</htdphtd:user>  
2188               <htdphtd:user>Karsten</htdphtd:user>  
2189               <htdphtd:user>Matthias</htdphtd:user>  
2190               <htdphtd:user>Patrick</htdphtd:user>  
2191             </htdphtd:users>  
2192           </htdphtd:organizationalEntity>  
2193         </htdphtc:potentialOwners>  
2194       </htdphtc:peopleAssignments>  
2195     </htdphtc:humanTaskContext>  
2196   </S:Header>  
2197   <S:Body>...</S:Body>  
2198 </S:Envelope>
```

- Formatted: German (Germany)

## 2199 7.5 Human Task Policy Assertion

2200 In order to support discovery of Web services that support the human task contract that are available for  
2201 coordination by another service, a *human task policy* assertion is defined by WS-HumanTask. This policy  
2202 assertion may be associated with the business operation used by the invoking component (recall that the  
2203 human task is restricted to have exactly one business operation). In doing so, the provider of a human  
2204 task may signal whether or not the corresponding task may communicate with an invoking component via  
2205 the WS-HT coordination protocol.

2206 The following describes the policy assertion used to specify that an operation can be used to instantiate a  
2207 human task with the proper protocol in place:

```
2208 | <htdph:HumanTaskAssertion wsp:Optional="true"? ...>  
2209 |   ...  
2210 | </htdph:HumanTaskAssertion>
```

```
2211 |  
2212 | /htdph:HumanTaskAssertion
```

2213 This policy assertion specifies that the sender of an input message MUST include context  
2214 information for a human task coordination type passed with the message. The receiving human  
2215 task MUST be instantiated with the WS-Human Task protocol in place.

```
2216 |  
2217 | /htdph:HumanTaskAssertion/@wsp:Optional="true"
```

2218 As defined in WS-Policy [WS-Policy], this is the compact notation for two policy alternatives, one  
2219 with and one without the assertion. Presence of both policy alternatives indicates that the  
2220 behavior indicated by the assertion is optional, such that a WS-HT coordination context MAY be  
2221 passed with an input message. If the context is passed the receiving human task MUST be  
2222 instantiated with the WS-HT protocol in place. The absence of the assertion is interpreted to  
2223 mean that a WS-HT coordination context SHOULD NOT be passed with an input message.

2224

2225 The human task policy assertion indicates behavior for a single operation, thus the assertion has an  
2226 Operation Policy Subject. WS-PolicyAttachment [WS-PolAtt] defines two policy attachment points with  
2227 Operation Policy Subject, namely wsdl:portType/wsdl:operation and wsdl:binding/wsdl:operation.

2228 | The <htdph:HumanTaskAssertion> policy assertion can also be used for notifications. In that case  
2229 it means that the sender of an input message MAY pass the human task context information with the  
2230 message. Other headers, including headers with the coordination context are ignored.

2231

## 8 Providing Callback Information for Human Tasks

2232 WS-HumanTask extends the information model of a WS-Addressing endpoint reference (EPR) defined in  
2233 [WS-Addr-Core] (see [WS-Addr-SOAP] and [WS-Addr-WSDL] for more details). This extension is needed  
2234 to support passing information to human tasks about ports and operations of a caller receiving responses  
2235 from such human tasks.

2236 Passing this callback information from a caller (i.e. a requesting application) to a human task may  
2237 override static deployment information that may have been set.

### 8.1 EPR Information Model Extension

2238 Besides the properties of an endpoint reference (EPR) defined by [WS-Addr-Core] WS-HumanTask  
2239 defines the following abstract properties:

2240

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

2241

2242 This property contains the value of the [action] message addressing property to be sent within the  
2243 response message.

2244

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

2245

2246 This property contains the name of a WSDL operation.

2247

2248 Each of these properties is a child element of the [metadata] property of an endpoint reference. An  
2249 endpoint reference passed by a caller to a human task MUST contain the [metadata] property.

2250 Furthermore, this [metadata] property MUST contain either a [response action] property or a [response  
2251 operation] property.

2252 If present, the value of the [response action] property MUST be used by the WS-HT implementation  
2253 hosting the responding human task to specify the value of the [action] message addressing property of  
2254 the response message sent back to the caller. Furthermore, the [destination] property of this response  
2255 message MUST be copied from the [address] property of the EPR contained in the original request  
2256 message.

2257 If present, the value of the [response operation] property MUST be the name of an operation of the port  
2258 type implemented by the endpoint denoted by the [address] property of the EPR. The corresponding port  
2259 type MUST be included as a WSDL 1.1 definition nested within the [metadata] property of the EPR (see  
2260 [WS-Addr-WSDL]). The WS-HT implementation hosting the responding human task MUST use the value  
2261 of the [response operation] property as operation of the specified port type at the specified endpoint to  
2262 send the response message. Furthermore, the [metadata] property MUST contain WSDL 1.1 binding  
2263 information corresponding to the port type implemented by the endpoint denoted by the [address]  
2264 property of the EPR.

2265 The EPR sent from the caller to the human task MUST identify the instance of the caller. This can be  
2266 done in two ways: First, the value of the [address] property may contain a URL with appropriate  
2267 parameters uniquely identifying the caller instance. Second, appropriate [reference parameters]  
2268 properties are specified within the EPR. The values of these [reference parameters] uniquely identify the  
2269 caller within the scope of the URI passed within the [address] property.

### 8.2 XML Infoset Representation

2270 The following describes the infoset representation of the EPR extensions introduced by WS-HumanTask:

```

2275 <wsa:EndpointReference>
2276   <wsa:Address>xsd:anyURI</wsa:Address>
2277   <wsa:ReferenceParameters>xsd:any*</wsa:ReferenceParameters?>
2278   <wsa:Metadata>
2279     <htcphtcp:responseAction>xsd:anyURI</htcphtcp:responseAction?>
2280     <htcphtcp:responseOperation>xsd:NCName</htcphtcp:responseOperation?>
2281   </wsa:Metadata>
2282 </wsa:EndpointReference>

```

Formatted: Portuguese (Brazil)

2283  
2284 */wsa:EndpointReference/wsa:Metadata*

2285 This is a MANDATORY element of the EPR sent. It MUST either contain WSDL 1.1 metadata  
2286 specifying the information to access the endpoint (i.e. its port type, bindings or ports) according to  
2287 [WS-Addr-WSDL] as well as a <htcp:responseOperation> element, or it MUST contain a  
2288 <htcp:responseAction> element.

2289 */wsa:EndpointReference/wsa:Metadata/htcphtcp:responseAction*

2290 This element (of type xsd:anyURI) specifies the value of the [action] message addressing  
2291 property to be used by the receiving human task when sending the response message from the  
2292 human task back to the caller. If this element is specified the <htcp:responseOperation>  
2293 element MUST NOT be specified.

2294 */wsa:EndpointReference/wsa:Metadata/htcphtcp:responseOperation*

2295 This element (of type xsd:NCName) specifies the name of the operation to be used by the  
2296 receiving human task to send the response message from the human task back to the caller. The  
2297 value of this element is taken from the htd:callRemoteTask/@responseOperation  
2298 attribute. If this element is specified the <htcp:responseAction> element MUST NOT be  
2299 specified.

2300 Effectively, WS-HumanTask defines two ways to pass callback information from the caller to the human  
2301 task. First, the EPR contains just the value of the [action] message addressing property to be used within  
2302 the response message (i.e. the <htcp:responseAction> element). Second, the EPR contains the  
2303 WSDL 1.1 metadata for the port receiving the response operation. In this case, the callback information  
2304 also specifies which operation of that port is to be used (i.e. the <htcp:responseOperation>  
2305 element). In both cases, the response is typically sent to the address specified in the <wsa:Address>  
2306 element of the EPR contained in the original request message; note, that [WS-Addr-WSDL] does not  
2307 exclude redirection to other addresses than the one specified, but the corresponding mechanisms are out  
2308 of the scope of the specification.

2309 The following example of an endpoint reference shows the usage of the <htcp:responseAction>  
2310 element. The <wsa:Metadata> elements contain the <htcp:responseAction> element that  
2311 specifies the value of the [action] message addressing property to be used by the WS-HT implementation  
2312 when sending the response message back to the caller. This value is  
2313 http://example.com/LoanApproval/approvalResponse. The value of the [destination] message  
2314 addressing property to be used is given in the <wsa:Address> element, namely  
2315 http://example.com/LoanApproval/loan?ID=42. Note that this URL includes the HTTP search  
2316 part with the parameter ID=42 which uniquely identifies the instance of the caller.

```

2317 <wsa:EndpointReference
2318   xmlns:wsa="http://www.w3.org/2005/08/addressing">
2319   <wsa:Address>http://example.com/LoanApproval/loan?ID=42</wsa:Address>
2320   <wsa:Metadata>
2321     <htcphtcp:responseAction>
2322       http://example.com/LoanApproval/approvalResponse
2323     </htcphtcp:responseAction>
2324   </wsa:Metadata>
2325 </wsa:EndpointReference>
2326
2327

```

Formatted: Portuguese (Brazil)

2328 </wsa:EndpointReference>

2329

2330 The following example of an endpoint reference shows the usage of the <http:responseOperation>  
2331 element and corresponding WSDL 1.1 metadata. The port type of the caller that receives the response  
2332 message from the WS-HT implementation is defined using the <wsdl:portType> element. In our  
2333 example it is the LoanApprovalPT port type. The definition of the port type is nested in a corresponding  
2334 WSDL 1.1 <wsdl:definitions> element in the <wsa:Metadata> element. This  
2335 <wsdl:definitions> element also contains a binding for this port type as well as a corresponding  
2336 port definition nested in a <wsdl:service> element. The <http:responseOperation> element  
2337 specifies that the approvalResponse operation of the LoanApprovalPT port type must be used to  
2338 send the response to the caller. The address of the actual port to be used which implements the  
2339 LoanApprovalPT port type and thus the approvalResponse operation is given in the  
2340 <wsa:Address> element, namely the URL http://example.com/LoanApproval/loan. The  
2341 unique identifier of the instance of the caller is specified in the <MyInstanceID<xmp:MyInstanceID>  
2342 element nested in the <wsa:ReferenceParameters> element.

```
2343 <wsa:EndpointReference
2344   xmlns:wsa="http://www.w3.org/2005/08/addressing">
2345
2346   <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>
2347
2348   <wsa:ReferenceParameters>
2349     <MyInstanceID<xmp:MyInstanceID>42</MyInstanceID>
2350   </wsa:ReferenceParameters>
2351
2352   <wsa:Metadata>
2353
2354     <wsdl:definitions ...>
2355
2356       <wsdl:portType name="LoanApprovalPT">
2357         <wsdl:operation name="approvalResponse">...</wsdl:operation>
2358         ...
2359       </wsdl:portType>
2360
2361       <wsdl:binding name="LoanApprovalSoap" type="LoanApprovalPT">
2362         ...
2363       </wsdl:binding>
2364
2365       <wsdl:service name="LoanApprovalService">
2366         <wsdl:port name="LA" binding="LoanApprovalSoap">
2367           <soap:address
2368             location="http://example.com/LoanApproval/loan" />
2369         </wsdl:port>
2370         ...
2371       </wsdl:service>
2372
2373     </wsdl:definitions>
2374
2375     <http:responseOperation>approvalResponse</http:responseOperation>
2376
2377   </wsa:Metadata>
2378 </wsa:EndpointReference>
```

## 2380 8.3 Message Addressing Properties

2381 Message addressing properties provide references for the endpoints involved in an interaction at the  
2382 message level. For this case, WS-HumanTask uses the message addressing properties defined in [WS-  
2383 Addr-Core] for the request message as well as for the response message.

2384 The request message sent by the caller (i.e. the requesting application) to the human task uses the  
2385 message addressing properties as described in [WS-Addr-Core]. WS-HumanTask refines the use of the  
2386 following message addressing properties:

- 2387 • The [reply endpoint] message addressing property MUST contain the EPR to be used by the  
2388 human task to send its response to.

2389

2390 Note that the [fault endpoint] property is not used by WS-HumanTask. This is because via one-way  
2391 operation no application level faults are returned to the caller.

2392 The response message sent by the human task to the caller uses the message addressing properties as  
2393 defined in [WS-Addr-Core] and refines the use of the following properties:

- 2394 • The value of the [action] message addressing property is set as follows:
  - 2395 • If the original request message contains the `<htcp:responseAction>` element in the  
2396 `<wsa:Metadata>` element of the EPR of the [reply endpoint] message addressing property,  
2397 the value of the former element is copied into the [action] property of the response message.
  - 2398 • If the original request message contains the `<htcp:responseOperation>` element (and,  
2399 thus, WSDL 1.1 metadata) in the `<wsa:Metadata>` element of the EPR of the [reply  
2400 endpoint] message addressing property, the value of the [action] message addressing  
2401 property of the response message is determined as follows:
    - 2402 • Assume that the WSDL 1.1 metadata specifies within the binding chosen a value for the  
2403 `soapaction` attribute on the `soap:operation` element of the response operation.  
2404 Then, this value MUST be used as value of the [action] property.
    - 2405 • If no such `soapaction` attribute is provided, the value of the [action] property MUST be  
2406 derived as specified in [WS-Addr-WSDL].
- 2407 • Reference parameters are mapped as specified in [WS-Addr-SOAP].

## 2408 8.4 SOAP Binding

2409 A SOAP binding specifies how abstract message addressing properties are bound to SOAP headers. In  
2410 this case, WS-HumanTask uses the mappings as specified by [WS-Addr-SOAP].

2411 The following is an example of a request message sent from the caller to the human task containing the  
2412 `<htcp:responseAction>` element in the incoming EPR. The EPR is mapped to SOAP header fields as  
2413 follows: The endpoint reference to be used by the human task for submitting its response message to is  
2414 contained in the `<wsa:ReplyTo>` element. The address of the endpoint is contained in the  
2415 `<wsa:Address>` element. The identifier of the instance of the caller to be encoded as reference  
2416 parameters in the response message is nested in the `<wsa:ReferenceParameters>` element. The  
2417 value of the `<wsa:Action>` element to be set by the human task in its response to the caller is in the  
2418 `<htcp:responseAction>` element nested in the `<wsa:Metadata>` element of the EPR.

```
2419 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"  
2420   xmlns:wsa="http://www.w3.org/2005/08/addressing"  
2421   xmlns:htdphtcp="http://www.example.org/WS-HT/protocolhttp://docs.oasis-  
2422   open.org/ns/bpel4people/ws-humantask/protocol/200803">  
2423  
2424   <S:Header>  
2425     <wsa:ReplyTo>  
2426       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>  
2427     <wsa:ReferenceParameters>
```

```

2428 |     <xyz:xmp:MyInstanceID>42</xyz:xmp:MyInstanceID>
2429 |   </wsa:ReferenceParameters>
2430 |   <wsa:Metadata>
2431 |     <htc:responseAction>
2432 |       http://example.com/LoanApproval/approvalResponse
2433 |     </htc:responseAction>
2434 |   </wsa:Metadata>
2435 | </wsa:ReplyTo>
2436 | </S:Header>
2437 |
2438 | <S:Body>...</S:Body>
2439 | </S:Envelope>
2440 |

```

2441 The following is an example of a response message corresponding to the request message discussed  
 2442 above. This response is sent from the human task back to the caller. The <wsa:To> element contains a  
 2443 copy of the <wsa:Address> element of the original request message. The <wsa:Action> element is  
 2444 copied from the <htc:responseAction> element of the original request message. The reference  
 2445 parameters are copied as standalone elements (the <xyz:xmp:MyInstanceID> element below) out of  
 2446 the <wsa:ReferenceParameters> element of the request message.

```

2447 | <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
2448 |   xmlns:wsa="http://www.w3.org/2005/08/addressing">
2449 |   <S:Header>
2450 |     <wsa:To>
2451 |       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>
2452 |     </wsa:To>
2453 |     <wsa:Action>
2454 |       http://example.com/LoanApproval/approvalResponse
2455 |     </wsa:Action>
2456 |     <xyz:xmp:MyInstanceID wsa:IsReferenceParameter='true'>
2457 |       42
2458 |     </xyz:xmp:MyInstanceID>
2459 |   </S:Header>
2460 |   <S:Body>...</S:Body>
2461 | </S:Envelope>
2462 |

```

2463 The following is an example of a request message sent from the caller to the human task containing the  
 2464 <htc:responseOperation> element and corresponding WSDL metadata in the incoming EPR. The  
 2465 EPR is mapped to SOAP header fields as follows: The endpoint reference to be used by the human task  
 2466 for submitting its response message to is contained in the <wsa:ReplyTo> element. The address of the  
 2467 endpoint is contained in the <wsa:Address> element. The identifier of the instance of the caller to be  
 2468 encoded as reference parameters in the response message is nested in the  
 2469 <wsa:ReferenceParameters> element. The WSDL metadata of the endpoint is contained in the  
 2470 <wsdl:definitions> element. The name of the operation of the endpoint to be used to send the  
 2471 response message to is contained in the <htc:responseOperation> element. Both elements are  
 2472 nested in the <wsa:Metadata> element of the EPR. These elements provide the basis to determine the  
 2473 value of the action header field to be set by the caller in its response to the caller.

```

2474 | <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
2475 |   xmlns:wsa="http://www.w3.org/2005/08/addressing"
2476 |   xmlns:htc="http://www.example.org/WS-HT/protocol"xmlns:open="http://docs.oasis-
2477 | open.org/ns/bpel4people/ws-humantask/protocol/200803">
2478 |   <S:Header>
2479 |     <wsa:ReplyTo>
2480 |
2481 |       <wsa:Address>http://example.com/LoanApproval/loan</wsa:Address>
2482 |

```

```

2483 <wsa:ReferenceParameters>
2484 <xyz:xmp:MyInstanceID>42</xyz:xmp:MyInstanceID>
2485 </wsa:ReferenceParameters>
2486
2487 <wsa:Metadata>
2488
2489 <wsdl:definitions
2490 targetNamespace="http://example.com/loanApproval"
2491 xmlns:wsdl="..." xmlns:soap="...">
2492
2493 <wsdl:portType name="LoanApprovalPT">
2494 <wsdl:operation name="approvalResponse">
2495 <wsdl:input name="approvalInput" ... />
2496 </wsdl:operation>
2497 ...
2498 </wsdl:portType>
2499
2500 <wsdl:binding name="LoanApprovalSoap"
2501 type="LoanApprovalPT">
2502 ...
2503 </wsdl:binding>
2504
2505 <wsdl:service name="LoanApprovalService">
2506 <wsdl:port name="LA" binding="LoanApprovalSoap">
2507 <soap:address
2508 location="http://example.com/LoanApproval/loan" />
2509 </wsdl:port>
2510 ...
2511 </wsdl:service>
2512 </wsdl:definitions>
2513
2514 <httcp:responseOperation>
2515 approvalResponse
2516 </httcp:responseOperation>
2517
2518 </wsa:Metadata>
2519 </wsa:ReplyTo>
2520
2521 </S:Header>
2522 <S:Body>...</S:Body>
2523 </S:Envelope>
2524

```

2525 The following is an example of a response message corresponding to the request message before; this
2526 response is sent from the human task back to the caller. The `<wsa:To>` element contains a copy of the
2527 `<wsa:Address>` field of the original request message. The reference parameters are copied as
2528 standalone element (the `<xyz:xmp:MyInstanceID>` element below) out of the
2529 `<httcp:ReferenceParameters>` element of the request message. The value of the `<wsa:Action>`
2530 element is composed according to [WS-Addr-WSDL] from the target namespace, port type name, name
2531 of the response operation to be used, and name of the input message of this operation given in the code
2532 snippet above.

```

2533 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
2534 xmlns:wsa="http://www.w3.org/2005/08/addressing"
2535 xmlns:htd="http://www.example.org/WS-HTTP" http://docs.oasis-
2536 open.org/ns/bpel4people/ws-humantask/200803">
2537 <S:Header>
2538 <wsa:To>http://example.com/LoanApproval/loan</wsa:To>
2539 <wsa:Action>

```

```
2540     http://example.com/loanApproval/...
2541     ...LoanApprovalPT/approvalResponse/ApprovalInput
2542     </wsa:Action>
2543     <xyz:xmp:MyInstanceID wsa:IsReferenceParameter='true'>
2544     42
2545     </xyz:xmp:MyInstanceID>
2546     </S:Header>
2547     <S:Body>...</S:Body>
2548     </S:Envelope>
```

2549

---

## 9 Security Considerations

2550 WS-HumanTask does not mandate the use of any specific mechanism or technology for client  
2551 authentication. However, a client MUST provide a principal or the principal MUST be obtainable by the  
2552 infrastructure.

2553 When using task APIs via SOAP bindings, compliance with the WS-I Basic Security Profile 1.0 is  
2554 RECOMMENDED.

---

2555 **10 Conformance**

2556 (tbd.)

2557

---

## 11 References

2558 [RFC 1766]

2559 Tags for the Identification of Languages, RFC 1766, available via  
2560 <http://www.ietf.org/rfc/rfc1766.txt>

2561 [RFC 2046]

2562 Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, RFC 2046, available via  
2563 <http://www.isi.edu/in-notes/rfc2046.txt> (or <http://www.iana.org/assignments/media-types/>)

2564 [RFC 2119]

2565 Key words for use in RFCs to Indicate Requirement Levels, RFC 2119, available via  
2566 <http://www.ietf.org/rfc/rfc2119.txt>

2567 [RFC 2396]

2568 Uniform Resource Identifiers (URI): Generic Syntax, RFC 2396, available via  
2569 <http://www.faqs.org/rfcs/rfc2396.html>

2570 [RFC 3066]

2571 Tags for the Identification of Languages, H. Alvestrand, IETF, January 2001, available via  
2572 <http://www.isi.edu/in-notes/rfc3066.txt>

2573 [WSDL 1.1]

2574 Web Services Description Language (WSDL) Version 1.1, W3C Note, available via  
2575 <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

2576 [WS-Addr-Core]

2577 Web Services Addressing 1.0 - Core, W3C Recommendation, May 2006, available via  
2578 <http://www.w3.org/TR/ws-addr-core>

2579 [WS-Addr-SOAP]

2580 Web Services Addressing 1.0 – SOAP Binding, W3C Recommendation, May 2006, available via  
2581 <http://www.w3.org/TR/ws-addr-soap>

2582 [WS-Addr-WSDL]

2583 Web Services Addressing 1.0 – WSDL Binding, W3C Working Draft, February 2006, available  
2584 via <http://www.w3.org/TR/ws-addr-wsdl>

2585 [WS-C]

2586 Web Services Coordination (WS-Coordination) Version 1.1, OASIS Committee Specification,  
2587 February 2007, available via <http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec/wstx-wscoor-1.1-spec.html>

2589 [WS-Policy]

2590 Web Services Policy 1.5 - Framework, W3C Candidate Recommendation 30 March 2007,  
2591 available via <http://www.w3.org/TR/ws-policy/>

2592 [WS-PolAtt]

2593 Web Services Policy 1.5 - Attachment, W3C Candidate Recommendation 30 March 2007,  
2594 available via <http://www.w3.org/TR/2007/CR-ws-policy-attach-20070330/>

2595 [XML Infoset]

2596 XML Information Set, W3C Recommendation, available via <http://www.w3.org/TR/2001/REC-xml-infoset-20011024/>

2598 [XML Namespaces]

- 2599           Namespaces in XML 1.0 (Second Edition), W3C Recommendation, available via  
2600           <http://www.w3.org/TR/REC-xml-names/>
- 2601   [XML Schema Part 1]
- 2602           XML Schema Part 1: Structures, W3C Recommendation, October 2004, available via  
2603           <http://www.w3.org/TR/xmlschema-1/>
- 2604   [XML Schema Part 2]
- 2605           XML Schema Part 2: Datatypes, W3C Recommendation, October 2004, available via  
2606           <http://www.w3.org/TR/xmlschema-2/>
- 2607   [XMLSpec]
- 2608           XML Specification, W3C Recommendation, February 1998, available via  
2609           <http://www.w3.org/TR/1998/REC-xml-19980210>
- 2610   [XPATH 1.0]
- 2611           XML Path Language (XPath) Version 1.0, W3C Recommendation, November 1999, available via  
2612           <http://www.w3.org/TR/1999/REC-xpath-19991116>

2613

## A. Portability and Interoperability Considerations

2614 This section illustrates the portability and interoperability aspects addressed by WS-HumanTask:

- 2615 • Portability - The ability to take human tasks and notifications created in one vendor's environment  
2616 and use them in another vendor's environment.
- 2617 • Interoperability - The capability for multiple components (task infrastructure, task list clients and  
2618 applications or processes with human interactions) to interact using well-defined messages and  
2619 protocols. This enables combining components from different vendors allowing seamless  
2620 execution.

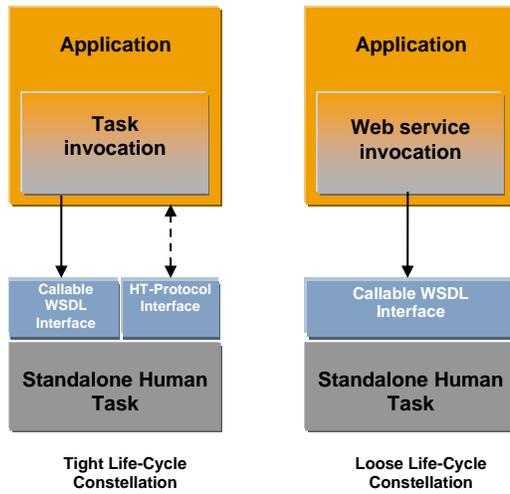
2621 Portability requires support of WS-HumanTask artifacts.

2622 Interoperability between task infrastructure and task list clients is achieved using the operations for client  
2623 applications.

2624 Interoperability between applications and task infrastructure from different vendors subsumes two  
2625 alternative constellations depending on how tightly the life-cycles of the task and the invoking  
2626 application are coupled with each other. This is shown in the figure below:

2627 Tight Life-Cycle Constellation: Applications are human task aware and control the life cycle of tasks.

2628 Interoperability between applications and task infrastructure is achieved using the WS-HumanTask  
2629 coordination protocol.



2630 Loose Life-Cycle Constellation: Applications use basic Web services protocols to invoke Web services  
2631 implemented as human tasks. In this case standard Web services interoperability is achieved and  
2632 applications do not control the life cycle of tasks.

2633

---

## B. WS-HumanTask Language Schema

2634  
2635

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

2636

`ws-humantask.xsd`

2637  
2638

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

2639

## C. WS-HumanTask Data Types Schema

2640

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

2641

2642

ws-humantask-types.xsd

2643

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

2644

Formatted: Bullets and Numbering

2645  
2646

## G.D. Operations-WS-HumanTask API WSDLPort Types

2647 Note to specification editors: the ~~WS-HumanTask API data types XML Schema definition, the WS-~~  
2648 ~~HumanTask API operation signature XML Schema definition, and the WS-HumanTask API WSDL~~  
2649 definition ~~are is~~ separately maintained in artifacts

2650 ~~ws-humantask-api.xsd~~

2651 ~~ws-humantask-api-wsdl.xsd~~

2652 ws-humantask-api.wsdl

2653 | The contents of ~~these~~ this artifact shall be copied back into this section before publishing the  
2654 specification, e.g., as a committee draft.

2655

## **E. WS-HumanTask Protocol Handler Port Types**

2656

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

2657

2658

[ws-humantask-protocol.wsdl](#)

2659

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

2660

2661

## **F. WS-HumanTask Context Schema**

2662

Note to specification editors: the WS-HumanTask context XML Schema definition is separately

2663

maintained in an artifact

2664

[ws-humantask-context.xsd](#)

2665

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

2666

as a committee draft.

2667

## **G. WS-HumanTask Policy Assertion Schema**

2668  
2669

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

2670

ws-humantask-policy.xsd

2671  
2672

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

2673 **D.H. Sample**

2674 This appendix contains the full sample used in this specification.

2675

2676 **WSDL Definition**

2677 Note to specification editors: the WS-HumanTask example WSDL definition is separately maintained in  
2678 an artifact

2679 `ws-humantask-example-claim-approval.wsdl`

2680 The contents of this artifact shall be copied back into this section before publishing the specification, e.g.,  
2681 as a committee draft.

2682

2683 **Human Interaction Definition**

2684 Note to specification editors: the WS-HumanTask example Human Task definition is separately  
2685 maintained in an artifact

2686 `ws-humantask-example-claim-approval.tel`

2687 The contents of this artifact shall be copied back into this section before publishing the specification, e.g.,  
2688 as a committee draft.

2689 **E.Schema of Protocol Messages**

2690 ~~Note to specification editors: the WS-HumanTask-protocol.XML Schema definition is separately~~  
2691 ~~maintained in an artifact~~

2692 ~~ws-humantask-protocol.xsd~~

2693 ~~The contents of this artifact shall be copied back into this section before publishing the specification, e.g.,~~  
2694 ~~as a committee draft.~~

2695  
2696  
2697  
2698  
2699  
2700

## 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.

2701  
2702  
2703  
2704  
2705  
2706

## ~~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.~~

2707

## H.I. Acknowledgements

2708 The following individuals have participated in the creation of this specification and are gratefully  
2709 acknowledged:

2710

### 2711 **Members of the BPEL4People Technical Committee:**

2712 Ashish Agrawal, Adobe Systems

2713 Mike Amend, BEA Systems, Inc.

2714 Stefan Baeuerle, SAP AG

2715 Charlton Barreto, Adobe Systems

2716 Justin Brunt, TIBCO Software Inc.

2717 Martin Chapman, Oracle Corporation

2718 James Bryce Clark, OASIS

2719 Luc Clément, Active Endpoints, Inc.

2720 Manoj Das, Oracle Corporation

2721 Mark Ford, Active Endpoints, Inc.

2722 Sabine Holz, SAP AG

2723 Dave Ings, IBM

2724 Gershon Janssen, Individual

2725 Diane Jordan, IBM

2726 Anish Karmarkar, Oracle Corporation

2727 Ulrich Keil, SAP AG

2728 Oliver Kieselbach, SAP AG

2729 Matthias Kloppmann, IBM

2730 Dieter König, IBM

2731 Marita Kruempelmann, SAP AG

2732 Frank Leymann, IBM

2733 Mark Little, Red Hat

2734 Ashok Malhotra, Oracle Corporation

2735 Mike Marin, IBM

2736 Mary McRae, OASIS

2737 Vinkesh Mehta, Deloitte Consulting LLP

2738 Jeff Mischkinsky, Oracle Corporation

2739 Ralf Mueller, Oracle Corporation

2740 Krasimir Nedkov, SAP AG

2741 Benjamin Notheis, SAP AG

2742 Michael Pellegrini, Active Endpoints, Inc.

2743 Gerhard Pfau, IBM

2744 Karsten Ploesser, SAP AG

2745 Ravi Rangaswamy, Oracle Corporation

2746 Alan Rickayzen, SAP AG

2747 Michael Rowley, BEA Systems, Inc.  
2748 Ron Ten-Hove, Sun Microsystems  
2749 Ivana Trickovic, SAP AG  
2750 Alessandro Triglia, OSS Nokalva  
2751 Claus von Riegen, SAP AG  
2752 Peter Walker, Sun Microsystems  
2753 Franz Weber, SAP AG  
2754 Prasad Yendluri, Software AG, Inc.  
2755

2756 **WS-HumanTask 1.0 Specification Contributors:**

2757 Ashish Agrawal, Adobe  
2758 Mike Amend, BEA  
2759 Manoj Das, Oracle  
2760 Mark Ford, Active Endpoints  
2761 Chris Keller, Active Endpoints  
2762 Matthias Kloppmann, IBM  
2763 Dieter König, IBM  
2764 Frank Leymann, IBM  
2765 Ralf Müller, Oracle  
2766 Gerhard Pfau, IBM  
2767 Karsten Plösser, SAP  
2768 Ravi Rangaswamy, Oracle  
2769 Alan Rickayzen, SAP  
2770 Michael Rowley, BEA  
2771 Patrick Schmidt, SAP  
2772 Ivana Trickovic, SAP  
2773 Alex Yiu, Oracle  
2774 Matthias Zeller, Adobe  
2775

Formatted: English (United States)

Formatted: English (United States)

Formatted: English (United States)

2776 The following individuals have provided valuable input into the design of this specification: Dave Ings,  
2777 Diane Jordan, Mohan Kamath, Ulrich Keil, Matthias Kruse, Kurt Lind, Jeff Mischkinsky, Bhagat Nainani,  
2778 Michael Pellegrini, Lars Rueter, Frank Ryan, David Shaffer, Will Stallard, Cyrille Waguët, Franz Weber,  
2779 and Eric Wittmann.

## H.J. Non-Normative Text

Formatted: Bullets and Numbering

2781

## **J.K.** Revision History

2782

[optional; should not be included in OASIS Standards]

2783

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
<a href="#">WD-02</a>	<a href="#">2008-06-25</a>	<a href="#">Ivana Trickovic</a>	<a href="#">Resolution of Issue #4 incorporated into the document/section 2.4.2</a>
<a href="#">WD-02</a>	<a href="#">2008-06-25</a>	<a href="#">Ivana Trickovic</a>	<a href="#">Resolution of Issue #4 incorporated into the ws-humantask.xsd</a>
<a href="#">WD-02</a>	<a href="#">2008-06-25</a>	<a href="#">Ivana Trickovic</a>	<a href="#">Resolution of Issue #8 incorporated into the document/section 6.2</a>
<a href="#">WD-02</a>	<a href="#">2008-06-25</a>	<a href="#">Ivana Trickovic</a>	<a href="#">Resolution of Issue #9 incorporated into the document/section 4.6 (example), and ws-humantask "ClaimApproval" example and WSDL file</a>
<a href="#">WD-02</a>	<a href="#">2008-06-28</a>	<a href="#">Dieter König</a>	<a href="#">Resolution of Issue #13 applied to complete document and all separate XML artifacts</a>
<a href="#">WD-02</a>	<a href="#">2008-06-28</a>	<a href="#">Dieter König</a>	<a href="#">Resolution of Issue #21 applied to section 2</a>
<a href="#">WD-02</a>	<a href="#">2008-07-08</a>	<a href="#">Ralf Mueller</a>	<a href="#">Resolution of Issue #14 applied to section 6, ws-humantask-api.wsdl and ws-humantask-types.xsd</a>
<a href="#">WD-02</a>	<a href="#">2008-07-15</a>	<a href="#">Luc Clément</a>	<a href="#">Updated Section 6.2 specifying (xsd:nonNegativeInteger) as the type for priority</a>

2784