# Web Services Context Specification (WS-Context)

**Committee draft version 0.8**

**Version created 3 November 2004**

**Editors**

Mark Little (mark.little@arjuna.com)

Eric Newcomer (eric.newcomer@iona.com)

Greg Pavlik (greg.pavlik@oracle.com)

# Abstract

Web services exchange XML documents with structured payloads. The processing semantics of an execution endpoint may be influenced by additional information that is defined at layers below the application protocol. When multiple Web services are used in combination, the ability to structure execution related data called context becomes important. This information is typically communicated via SOAP Headers. WS-Context provides a definition, a structuring mechanism, and a software service definition for organizing and sharing context across multiple execution endpoints.

The ability to compose arbitrary units of work is a requirement in a variety of aspects of distributed applications such as workflow and business-to-business interactions. By composing work, we mean that it is possible for participants in an activity to be able to determine unambiguously whether or not they are participating in the same activity.

An activity is the execution of multiple Web services composed using some mechanism external to this specification, such as an orchestration or choreography. A common mechanism is needed to capture and manage contextual execution environment data shared, typically persistently, across execution instances.

In order to correlate the work of participants within the same activity, it is necessary to propagate context to each participant. The context contains information (such as a unique identifier) that allows a series of operations to share a common outcome.

# Table of contents

# Note on terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 [2].

Namespace URIs of the general form "some-URI" represents some application-dependent or context-dependent URI as defined in RFC 2396 [3].

## 1.1 Namespace

The XML namespace URI that MUST be used by implementations of this specification is:

```
http://docs.oasis-open.org/wscaf/2004/09/wsctx
```

### 1.1.1 Prefix Namespace

| Prefix | Namespace |
|---|---|
| wsctx | **http://docs.oasis-open.org/wscaf/2004/09/wsctx** |
| ref | **http://docs.oasisopen.org/wsrm/2004/06/reference-1.1** |
| wsdl | **http://schemas.xmlsoap.org/wsdl/** |
| xsd | **http://www.w3.org/2001/XMLSchema** |
| wsu | **http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd** |
| tns | **targetNamespace** |

## 1.2 Referencing Specifications

One or more other specifications, such as (but not limited to) WS-CF may reference the WS-Context specification. The usage of optional items in WS-Context is typically determined by the requirements of such as referencing specification.

Referencing specifications are generally used to construct concrete protocols based on WS-Context. Any application that uses WS-Context must also decide what optional features are required. For the purpose of this document, the term *referencing specification* covers both formal specifications and more general applications that use WS-Context.

# 2 Architecture

An activity represents the execution of a series of related interactions with a set of Web Services; these operations are related via context. An activity is a conceptual grouping of services cooperating to perform some work; a context is the concrete manner in which this grouping occurs. The notion of an activity is used to scope application specific work. The definition of precisely what an activity is and what services it will require in order to perform that work, will depend upon the execution environment and application in which it is used.

Context contains information about the execution environment of an activity that supplements information in application payloads. Management of the basic context type is facilitated by services defined in this specification. The specification also provides service interfaces for managing session-oriented protocols and representing the corresponding activities with contexts. The overall architecture of the context is hierarchical and decomposable, e.g., it is possible to use the context structure without reference to any activity model.

The first element of the WS-Context specification is the context structure. The context structure defines a normal model for organizing context information. It supports nesting structures (parent-child relationships) for related contexts, and mechanisms to pass context information by reference or by value. A single context type is not sufficient for all applications; it must be extensible in a manner specific to a referencing specification and Web services must be able to augment the context, as they require.

WS-Context defines a *Context Service* for the management of activity contexts. The Context Service defines the scope of an activity and how information about it (the context) can be referenced and propagated in a distributed environment. The Context Service uses context to express basic information about the activity. The context is identified using a URI. The context contains information necessary for multiple Web services to be associated with the same activity. This information MAY be augmented when the context is created (by implementations of referencing specifications), or dynamically by application services as they send and receive contexts. Activities are represented by the Context Service, which maintains a repository of shared contexts. Whenever messages are exchanged within the scope of an activity, the Context Service can supply the associated context that MAY then be propagated with those messages.

Contexts MAY be passed by value (all of the information required to use the context is present in the data structure) or MAY be passed by reference (only a subset of the information is present in the data structure and the rest must be obtained by the receiving service). In order to support pass-by-reference, WS-Context defines an optional Context Manager Service that can be interrogated by a recipient of a reference context to obtain the contents of the context. This Context Manager Service MAY be the same as the Context Service, but there is no requirement for this within WS-Context.

## 2.1 Invocation of Service Operations

How application services are invoked is outside the scope of this specification: they MAY use synchronous or asynchronous message passing.

Irrespective of how remote invocations occur, context information related to the sender's activity needs to be referenced or propagated. This specification determines the format of the context, how it is referenced, and how a context may be created.

In order to support both synchronous and asynchronous interactions, the components are described in terms of the behavior and the interactions that occur between them. All interactions are described in terms of correlated messages, which a referencing specification MAY abstract at a higher level into request/response pairs.

Faults and errors that may occur when a service is invoked are communicated back to other Web services in the activity via SOAP messages that are part of the standard protocol. The fault mechanism of the underlying SOAP-based transport isn't used. For example, if an operation fails because no activity is present when one is required, then it will be valid for the InvalidContextFault message to be received by the response service. To accommodate other errors or faults, all response service signatures have a generalFault operation.

Note: in the rest of this specification we will use the term "invokes operation X on service Y" when referring to invoking services. This term does not imply a specific implementation for performing such service invocations and is used merely as shorthand for "sends message X to service Y." As long as implementations ensure that the on-the-wire message formats are compliant with those defined in this specification, how the end-points are implemented and how they expose the various operations (e.g., via WSDL [1]) is not mandated by this specification. However, a normative WSDL binding is provided by default in this specification.

> Note, this specification does not assume that a reliable message delivery mechanism has to be used for message interactions. As such, it MAY be implementation dependant as to what action is taken if a message is not delivered or no response is received.

## 2.2 Relationship to WSDL

Where WSDL is used in this specification it uses one-way messages with callbacks. This is the normative style. Other binding styles are possible (perhaps defined by referencing specifications), although they may have different acknowledgment styles and delivery mechanisms. It is beyond the scope of WS-Context to define these styles.

For clarity WSDL is shown in an abbreviated form in the main body of the document: only portTypes are illustrated; a default binding to SOAP 1.1-over-HTTP is also assumed as per [1].

## 2.3 Referencing and addressing conventions

There are multiple mechanisms for addressing messages and referencing Web services currently proposed by the Web services community. This specification defers the rules for addressing SOAP messages to existing specifications; the addressing information is assumed to be placed in SOAP headers and respect the normative rules required by existing specifications.

However, the Context message set requires an interoperable mechanism for referencing Web Services. For example, context structures may reference the service that is used to manage the content of the context. To support this requirement, WS-CAF has adopted an open content model for service references as defined by the Web Services Reliable Messaging Technical Committee [5]. The schema is defined in [6][7] and is shown in Figure 1.

```
<xsd:schema targetNamespace="http://docs.oasis-
open.org/wsrm/2004/06/reference-1.1.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.1">
  <xsd:complexType name="ServiceRefType">
    <xsd:sequence>
      <xsd:any namespace="##other" processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="reference-scheme" type="xsd:anyURI"
use="optional" />
  </xsd:complexType
```

Figure 1, service-ref Element

The ServiceRefType is extended by elements of the context structure as shown in Figure 2.

```
<xsd:element name=”context-manager” type=”ref:ServiceRefType”/>
```

Figure 2, ServiceRefType example.

Within the ServiceRefType, the reference-scheme is the namespace URI for the referenced addressing specification. For example, the value for WSRef defined in the WS-MessageDelivery specification [4] would be http://www.w3.org/2004/04/ws-messagedelivery. The value for WSRef defined in the WS-Addressing specification [8] would be http://schemas.xmlsoap.org/ws/2004/08/addressing. The reference scheme is optional and need only be used if the namespace URI of the QName of the Web service reference cannot be used to unambiguously identify the addressing specification in which it is defined.

The contents of the `xsd:any` element contain a service reference as defined by the referenced addressing specification. For example, a WSRef to a Context Manager Service would appear as shown in Figure 3.

```
<wsdl:service name="MyContextManager"
wsmd:portType="ctx:ContextManager">
<wsdl:port name="myCtxPort" binding="ex:ctxServiceBinding">
<soapbind:address location="http://example.com/wsdl-example1/impl"/>
</wsdl:port>
</wsdl:service>
```

Figure 3, WSRef to a Context Manager service.

Figure 4 illustrates how an element derived from the ServiceRefType can be used as a container for a WSRef.

```
<ctx:context-manager reference-scheme="http://www.w3.org/2004/04/ws-
messagedelivery">
    <wsdl:service name="MyContextService"
wsmd:portType="ctx:ContextManager">
        <wsdl:port name="myCtxPort" binding="ex:ctxServiceBinding">
            <soapbind:address location="http://example.com/wsdl-
example1/impl"/>
        </wsdl:port>
    </wsdl:service>
</ctx:service-ref>
```

Figure 4, example of a service-ref element

Messages sent to referenced services MUST use the addressing scheme defined by the specification indicated by the value of the reference-scheme element if present. Otherwise, the namespace URI associated with the Web service reference element MUST be used to determine the required addressing scheme. A service that requires a service reference element MUST use the mustUnderstand attribute for the SOAP header element within which it is enclosed and MUST return a mustUnderstand SOAP fault if the reference element isn't present and understood.

> Note, it is assumed that the addressing mechanism used by a given implementation supports a reply-to or sender field on each received message so that any required responses can be sent to a suitable response endpoint. This specification requires such support and does not define how responses are handled.

To preserve interoperability in deployments that contain multiple addressing schemes, there are no restrictions on a system, beyond those of the composite services themselves. However, it is RECOMMENDED where possible that composite applications confine themselves to the use of single addressing and reference model.

Because the prescriptive interaction pattern used by WS-Context is based on one-way messages with callbacks, it is possible that an endpoint may receive an unsolicited or unexpected message. The recipient is free to do whatever it wants with such messages.

# 3 Context

Context is used to include protocol specific data for transmission, typically (though not exclusively) in SOAP headers. The basic context structure is shown in Figure 5.

```
<xsd:complexType name="ContextType">
  <xsd:sequence>
    <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element name="context-identifier" type="
tns:contextIdentifierType xsd:"/>
    <xsd:element name="context-service" type="ref:ServiceRefType"
minOccurs="0"/>
    <xsd:element name="type" type="xsd:anyURI"/>
   <xsd:element name=”context-manager” type=”ref:ServiceRefType”
minOccurs=”0”/>
    <xsd:element name="parent-context" type=”tns:ContextType”
minOccurs="0">
  </xsd:sequence>
  <xsd:attribute name="timeout" type="xsd:int" use="optional"/>
  <xsd:attribute ref=”wsu:Id” use=”optional”/>
</xsd:complexType>
```

*Figure 5, Context Service Context.*

The context structure reflects some linear portion of a potentially tree-like relationship between contexts of the same type from the leaf to the root.

The context consists of the following items:

- A mandatory contextIdentifierType called *context-identifier*. This identifier can be thought of as a “correlation” identifier or a value that is used to indicate that a Web service is part of the same activity. The contextIdentifierType is a URI with an optional wsu:Id attribuite. It MUST be unique.
- An OPTIONAL ServiceRefType element, *context-service*, which identifies the issuing authority responsible for generating the context.
- A mandatory URI (*type*) that indicates the type of the protocol supported by the context, i.e., it indicates how the information within a context MAY be interpreted. This element MUST NOT be dereferenced and is only for unique identification. It MUST be unique amongst the referencing specifications.
- An OPTIONAL *context-manager* ServiceRefType to get data associated with a context-identifier that resolves to a reference to a Context Manager Web service. The presence of this reference indicates that the context has been passed by reference and it MAY be used to obtain the full value of the context later.
- An OPTIONAL *parent-contexts* element containing some portion of the current context’s parent hierarchy.
- An OPTIONAL *timeout* attribute, which indicates for how long the context information is valid; after this period has elapsed, the context is considered to be invalid. A context is determined to be valid by its issuing authority. For example, the WS-Context specification defines an issuing authority called the Context Service. The timeout allows the issuing authority implementation to invalidate contexts automatically rather than have them remain valid forever. It is implementation dependant as to the interpretation of a context with no specified timeout value.
- An OPTIONAL wsu:Id attribute, which may be used to support signing or encrypting the context structure.

- The context MAY contain information from an arbitrary number of augmenter services. The context structure is extended via the extensibility xsd:any element present in the schema for the ContextType.

Context propagation is possible using different protocols than those used by the application, as shown in Figure 6. The WS-Context specification does not assume a specific means by which contexts are associated with application messages, leaving this up to the referencing specification.

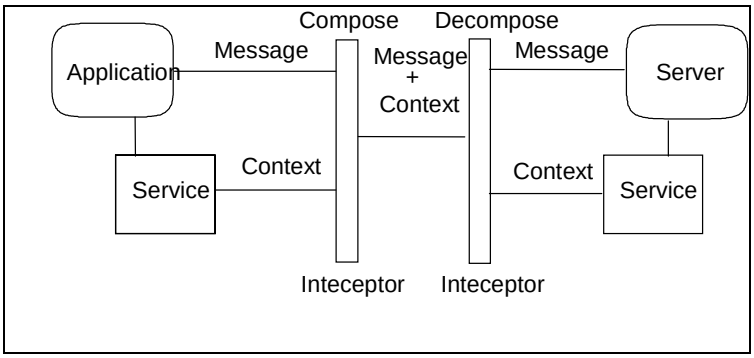


Figure 6, Services and context flow.

If a context is present on a received message and it contains a context-manager element then that element MAY be used by the recipient to dereference the context. By *dereference* we simply mean use the context-manager Web service to obtain the context. Any other information present in the received context at this point CANNOT be assumed to represent the current or entire contents of the context. If the context-manager is dereferenced, it MUST return the entire current contents of the context, i.e. the values corresponding to the context's ContextType elements held by the context service at the point of receiving the dereference message.

At a minimum, a context that is propagated by reference need only contain the context-identifier, type and context-manager elements. A context that is always propagated by value SHOULD NOT contain a context-manager element. A service that receives less than the minimum context MUST return a mustUnderstand exception when the mustUnderstand attribute is present.

> Note, if a referencing specification allows a context passed by reference to be updated at the context-manager, then a service that maintains a copy of a context which is passed by reference CANNOT assume that the cached copy is current.

The choice of whether to transmit a full or abbreviated context is left to the sender of the context. It is however expected that when dealing with large context elements that by-reference form will be used for efficiency. A sender who wishes to switch between full and abbreviated has the responsibility for ensuring that the dereferencing capability is available.

## 3.1 Activities

As mentioned earlier, an activity is defined as a collection of Web service operation invocations performed within a valid context. An activity is created, runs, and then completes. An outcome is the result of a completed activity. The expected semantics of a web service within an activity are defined by specifications derived from WS-Context. These semantics are indicated in a context by a protocol identifier representing the protocol type of the activity. The activity itself is uniquely identified by a context-identifier element.

In a system, there may be a set of contexts C associated with an activity. There will typically be multiple contexts because context data structures may be copied by value from service to service and may be augmented to include data that is valid to the local execution environment. The contexts in C are not equivalent: each may reflect one service's view of the

activity at a point in time. The initial context created for a specific activity is the base from which all other contexts may be derived.

A context is associated with one and only one activity; "compound" activity contexts do not exist, although nesting of activities MAY be supported. The set of operations represented by A may be used to define more than one activity; for example, the operations in A may include a context for a security protocol and a context for a transaction protocol, each representing a separate activity.

A Web service that performs an operation within an invalid context creates an invalid activity. It is up to the specifications using WS-Context to determine the implications of invalid activities (which may be insignificant or severe) and provide structuring mechanisms that avoid invalid activities if necessary.

Activities MAY be nested. If an activity is nested, then the global context MAY contain a hierarchy representing the activity structure. Each element in the context hierarchy MAY also possess a different context-identifier.

## 3.2 Context information and SOAP

Where messages (either application messages, or WS-Context protocol messages themselves) require contextualization, the context is transported in a SOAP header block. Referencing specifications determine if WS-Context actors must understand contexts that arrive in SOAP header blocks. In the example shown in Figure 7, the context propagated with application messages must be understood by their recipients. Hence in this case each SOAP header block carrying a context has the “mustUnderstand” attribute set to “true” (“1”) and the recipient must understand the header block encoding according to its identifying type URI.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2002/06/soap-envelope">
  <soap:Header>
    <context xmlns="http://docs.oasis-open.org/wscaf/2004/09/wsctx"
      timeout="100"
xmlns:wsdl=”http://schemas.xmlsoap.org/wsdl/”
xmlns:soapbind=http://schemas.xmlsoap.org/wsdl/soap/
soap:mustUnderstand="1">
      <context-identifier>
        http://docs.oasis-open.org/wscaf/2004/09/wsctx/abcdef:012345
      </context-identifier>
      <context-service>
        http://docs.oasis-open.org/wscaf/2004/09//wsctx/service
      </context-service>
      <type>
        http://docs.oasis-open.org/wscaf/2004/09/wsctx/context/type1
      </type>
      <parent-context>
          <timeout="200"/>
          <context-identifier>
            http://docs.oasis-
open.org/wscaf/2004/09//wsctx/5e4f2218b
          </context-identifier>
         <context-service>
           http://docs.oasis-open.org/wscaf/2004/09//wsctx/service
         </context-service>
       <type>
http://docs.oasis-open.org/wscaf/2004/09//wsctx/context/type1
       </type>
    </parent-context>
</context>
  </soap:Header>
  <soap:Body>
    <!-- Application Payload -->
  </soap:Body>
</soap:Envelope>
```

Figure 7, CTX Context Transported in a SOAP Header Block.

# 4 Context Manager

If the context is passed by reference, then a receiver may require the information it contains. WS-Context defines an OPTIONAL Web service interface, the Context Manager, which allows applications to retrieve and set data associated with a context. The Context Manager is only implemented to support contexts that are passed by reference. It is this Context Manager that is referenced by the presence of a context-manager element in a propagated context. Figure 8 shows the message interactions for the context using the dereferencing call-back style mentioned earlier: solid lines represent the initial request invocations and dashed lines represent the response invocations.

> Note, the Context Manager need not be the same endpoint as the Context Service.

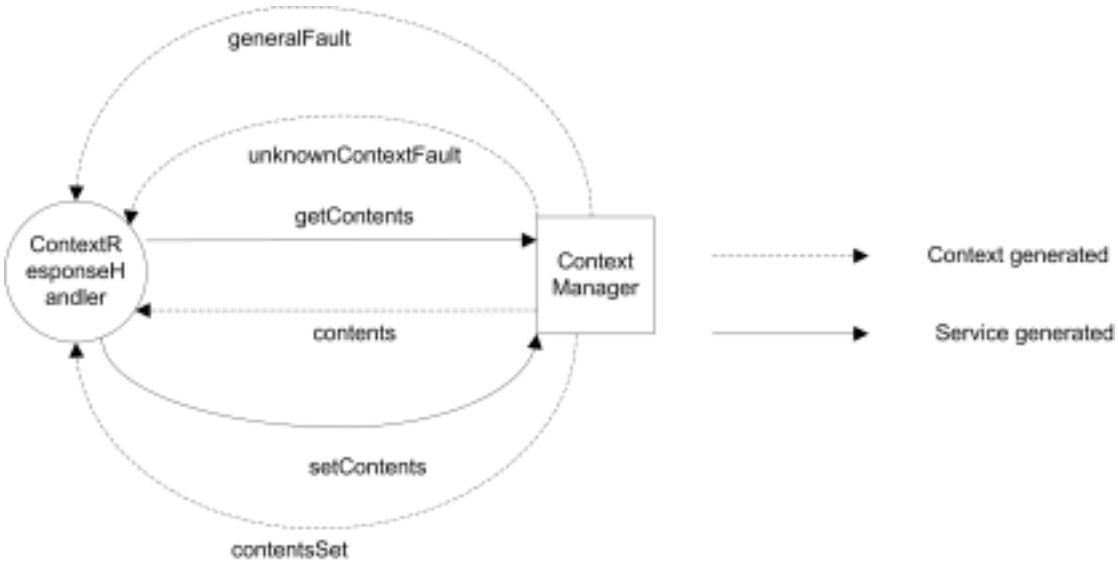


*Figure 8, Context interactions.*

The ContextManager has the following operations, all of which contain the callback address for the ContextResponseHandler:

- *getContents*: this message is used to request the entire contents of a specific context. It responds with either the *contents* message or an appropriate fault message. The entire contents of the context MUST be returned, i.e. the values corresponding to the context's ContextType elements.
- *setContents*: the contents of the context are replaced with the context information provided. It responds with either the *contentsSet* message or an appropriate fault message. Note that concurrency control of a context passed by a reference is an implementation issue.

  > Note, if the context is passed by reference and updates to it are allowed by the referencing specification, then some form of concurrency control protocol MAY be required to ensure that multiple updates do not conflict. It is implementation dependant as to what (or if) concurrency control is provided by the ContextManager.

The ContextResponseHandler has the following operations, all of which MUST be contextualized with at least a minimal context header, i.e., the context identifier:

- *contents*: this message is a response to *getContents* and returns the entire contents of a specific context.
- *contentsSet*: this message is sent as a response to *setContents* to indicate that contents of the context have been updated.
- *unknownContextFault*: this message is sent to indicate that the specified context cannot be located.
- *generalFault*: this message is sent to indicate that some other error has occurred during the enlistment.

The WSDL interfaces that elucidate these roles are shown in Figure 9.

```
<wsdl:portType name="ContextManagerPortType">
  <wsdl:operation name="getContents">
    <wsdl:input message="tns:GetContentsMessage"/>
  </wsdl:operation>
  <wsdl:operation name="setContents">
    <wsdl:input message="tns:SetContentsMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="ContextResponseHandlerPortType">
  <wsdl:operation name="contents">
    <wsdl:input message="tns:ContentsMessage"/>
  </wsdl:operation>
  <wsdl:operation name="contentsSet">
    <wsdl:input message="tns:ContentsSetMessage"/>
  </wsdl:operation>
  <wsdl:operation name="unknownContextFault">
    <wsdl:input message="tns:UnknownContextFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="generalFault">
    <wsdl:input message="tns:GeneralFaultMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

Figure 9, WSDL Interfaces for ContextManager and ContextResponseHandler Roles.

# 5 Context Service

The WS-Context specification defines a Context Service that supports the abstract notion of an activity and allows referencing specifications and services to scope work within these activities by sharing context. The basic infrastructure supports the lifecycle of contexts and ensures that each is uniquely identified. This section specifies how activities and contexts are modeled, managed, and represented by the Context Service.

## 5.1 Status

During the existence of the activity its status will either be running, completing, or completed. An activity SHOULD report its current status when asked; there is no notion of automatically informing services when a specific state is entered:

```
   <xsd:complexType name="StatusType">
          <xsd:simpleContent>
                 <xsd:extension base="xsd:string">
                        <xsd:attribute name="namespace"
type="xsd:anyURI"/>
                 </xsd:extension>
          </xsd:simpleContent>
   </xsd:complexType>
```

The namespace attribute is used by referencing specifications to qualify the value of the status string.

The WS-Context specification provides some basic status values. Referencing specifications may extend the basic status types provided by the WS-Context specification. The meaning of each of the string values is given below:

- activity.status.ACTIVE: The activity is in the active state. An implementation returns this status after an Activity has been started and prior to its beginning completion.
- activity.status.COMPLETING: The activity is in the process of completing. An implementation returns this status if it has started to complete, but has not yet finished the process. This value indicates that the activity may be performing activity specific work required to determine its final completion status, such as notifying participants of a failure. An activity MUST enter this state prior to completion.
- activity.status.COMPLETED: The activity has completed.
- activity.status.NO_ACTIVITY: There is no such activity.
- activity.status.UNKNOWN: The Context Service cannot determine the current status of the activity. This is a transient condition, and a subsequent invocation should ultimately return a different status. An implementation may attempt to retry the appropriate invocation transparently if such a value is returned initially.

The diagram below indicates the transitions that an Activity can undergo.

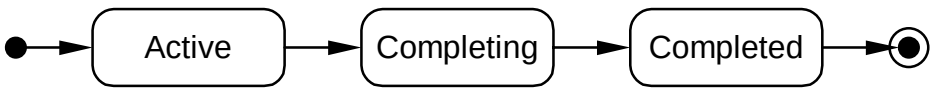


*Figure 10, Activity UML state diagram.*

## 5.2 Context Service messages

In order to be able to scope work within activities it is necessary for a component of the Context Service to provide an interface for activity demarcation. Since the Context Service maintains information on multiple activities, an activity context MAY be present on some operation invocations to determine the appropriate activity on which to operate. This context SHOULD be passed by reference, since it is only required for identification purposes.

Interactions with the Context Service occur between users (services) and the Context Service via the UserCTXService and CTXService interfaces respectively. The WSDL for these services is shown below and we shall describe the interactions in the following section.

```
<wsdl:portType name="CTXServicePortType">
  <wsdl:operation name="begin">
    <wsdl:input message="tns:BeginMessage"/>
  </wsdl:operation>
  <wsdl:operation name="complete">
    <wsdl:input message="tns:CompleteMessage"/>
  </wsdl:operation>
  <wsdl:operation name="getStatus">
    <wsdl:input message="tns:GetStatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="setTimeout">
    <wsdl:input message="tns:SetTimeoutMessage"/>
  </wsdl:operation>
  <wsdl:operation name="getTimeout">
    <wsdl:input message="tns:GetTimeoutMessage"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="UserCTXServicePortType">
  <wsdl:operation name="begun">
    <wsdl:input message="tns:BegunMessage"/>
  </wsdl:operation>
  <wsdl:operation name="completed">
    <wsdl:input message="tns:CompletedMessage"/>
  </wsdl:operation>
  <wsdl:operation name="status">
    <wsdl:input message="tns:StatusMessage"/>
  </wsdl:operation>
  <wsdl:operation name="requestedContext">
    <wsdl:input message="tns:RequestedContextMessage"/>
  </wsdl:operation>
  <wsdl:operation name="timeoutSet">
    <wsdl:input message="tns:TimeoutSetMessage"/>
  </wsdl:operation>
  <wsdl:operation name="timeout">
    <wsdl:input message="tns:TimeoutMessage"/>
  </wsdl:operation>
  <wsdl:operation name="invalidStateFault">
    <wsdl:input message="tns:InvalidStateFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="invalidContextFault">
    <wsdl:input message="tns:InvalidActivityFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="timeoutOutOfRange">
    <wsdl:input message="tns:TimeoutOutOfRangeFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="childActivityPendingFault">
    <wsdl:input message="tns:ChildActivityPendingFaultMessage"/>
  </wsdl:operation>
  <wsdl:operation name="noPermissionFault">
    <wsdl:input message="tns:NoPermissionFaultMessage"/>
  </wsdl:operation>
 </wsdl:portType>
```

*Figure 11, CTXService WSDL.*

In order to drive the Context Service, the following two roles (and associated services) are defined for the interactions:

- CTXService: this has operations begin, complete, getStatus, setTimeout and getTimeout;
- UserCTXService: this is the user/service callback endpoint address for the various CTXService operations. As such, it has operations begun, completed, status, timeoutSet, timeout, invalidStateFault, invalidActivityFault, timeoutOutOfRangeFault, childActivityPendingFault, noActivityFault, noPermissionFault, validContextExpectedFault.

The CTXService has the following operations, all of which are associated with the current context (if any). It is assumed that responses to these messages will be sent back using information present in whatever addressing scheme is used.

## begin

The begin operation creates a new context (based on the type parameter). If a context is present on the begin message then the new context is automatically nested with that context in a parent-child relationship, i.e., the propagated context is the immediate parent in the parent-contexts element, which MUST be set in the returned context.

> Note, it is not necessary for the entire parent-context hierarchy to be represented in the context structure. Some implementations and referencing specifications MAY wish to restrict this structure to only some linear subset of the hierarchy.

*begin* is therefore the first operation in an activity to use WS-Context. A unique context identifier is created for the context such that any context information that is subsequently obtained will reference this identifier. If a context is present on the begin request then the newly created context will be nested within it. Otherwise, the context exists at the top level. If the activity is completing, or has completed, the invalidContextFault operation will be invoked on the received UserCTXService endpoint.

If nesting of activities is not supported by the implementation then invalidActivityFault will be returned to the UserCTXService endpoint.

The timeout parameter is used to control the lifetime of a context. If the Activity has not completed by the time the timeout seconds elapse then it is subject to being completed automatically by the Context Service. The timeout can have the following possible values:

- *any positive value*: the Activity MUST complete within this number of seconds.
- -1: the Activity will never be completed automatically by the Context Service implementation, i.e., it will never be considered to have timed out.
- *0*: the last value specified using the set_timeout method is used. If no prior call to the *setTimeout* operation has occurred for this thread, or the value returned is 0, then it is implementation dependant as to the timeout value associated with this Activity.

Any other value results in the Context Service calling the timeoutOutOfRangeFault operation on the UserCTXService endpoint.

Upon success, the *begun* response will be sent and the new activity will be considered to be in the activity.status.ACTIVE state and the Context Service will invoke the begun operation of the UserCTXService.

If an invalid context is propagated on the begin request then the validContextExpectedFault operation is invoked on the UserCTXService.

The generalFault operation is invoked on the UserCTXService if an unexpected error or fault occurs.

The invalidProtocolTypeFault operation is invoked on the UserCTXService is the service cannot create a context of the required type.

## complete

A valid activity context is associated with this invocation. A Context Service implementation MAY impose restrictions on which Web services can terminate an activity, and in which case the noPermissionFault operation MAY be invoked on the UserCTXService. It is beyond the scope of this specification to determine how restrictions are imposed.

A protocol-specific completion command MAY accompany this invocation and MAY be used by the ContextService when terminating the activity. For example, one completion status for a transaction protocol might represent an abort signal. Some protocols may not make distinctions between success or failure in the termination of an activity and would not require any completion status.

Once complete, the Context Service sends the *completed* message to the UserCTXService. If the Activity has begun completion, or has completed, then the invalidContextFault UserCTXService operation is called.

If an invalid context is propagated on the request then the validContextExpectedFault operation is invoked on the UserCTXService.

The generalFault operation is invoked on the UserCTXService if an unexpected error or fault occurs.

## getStatus

This operation is used to obtain the current status of the activity referenced in the propagated context. The Context Service invokes the *status* operation on the associated UserCTXService to return the current status of the Activity. If there is no valid context associated with the context-identifier, the InvalidContextFault operation is invoked on the UserCTXService.

If an invalid context is propagated on the request then the validContextExpectedFault operation is invoked on the UserCTXService.

The generalFault operation is invoked on the UserCTXService if an unexpected error or fault occurs.

## setTimeout

No context is associated with this invocation. This operation modifies a state variable associated with the Context Service that affects the time-out period associated with the activities created by subsequent invocations of the begin operation when no timeout is specified (i.e., the begin timeout value is 0): this is a default timeout value associated with the service. If the parameter has a non-zero value n, then activities created by subsequent invocations of begin will be subject to being completed if they do not complete before n seconds after their creation. The timeout can have the following possible values:

- *any positive value*: the Activity MUST complete within this number of seconds.
- -1: the Activity will never be completed automatically by the Context Service implementation, i.e., it will never be considered to have timed out.
- *0*: it is implementation dependant as to the meaning of passing 0 as the value.

A valid timeout value results in the Context Service calling the UserCTXService's *timeoutSet* operation. Any other value results in the timeoutOutOfRangeFault operation being invoked on the associated UserCTXService.

The generalFault operation is invoked on the UserCTXService if an unexpected error or fault occurs.

## getTimeout

No context is associated with this invocation. Upon successful execution, this operation causes the Context Service to return the default timeout value (via the *timeout* message) associated with the service, i.e., the timeout that is associated with activities created by calls to begin when no timeout value is passed via begin. This need not be the value associated with the current Activity, however.

The generalFault operation is invoked on the UserCTXService if an unexpected error or fault occurs.

## 5.2.1 State transitions

Figure 12 shows the state transitions for an activity and how they relate to the various message exchanges between the client/user of the Context Service and the Context Service. As mentioned above, in order to participate in these message interactions the client/user supplies a UserCTXService endpoint.

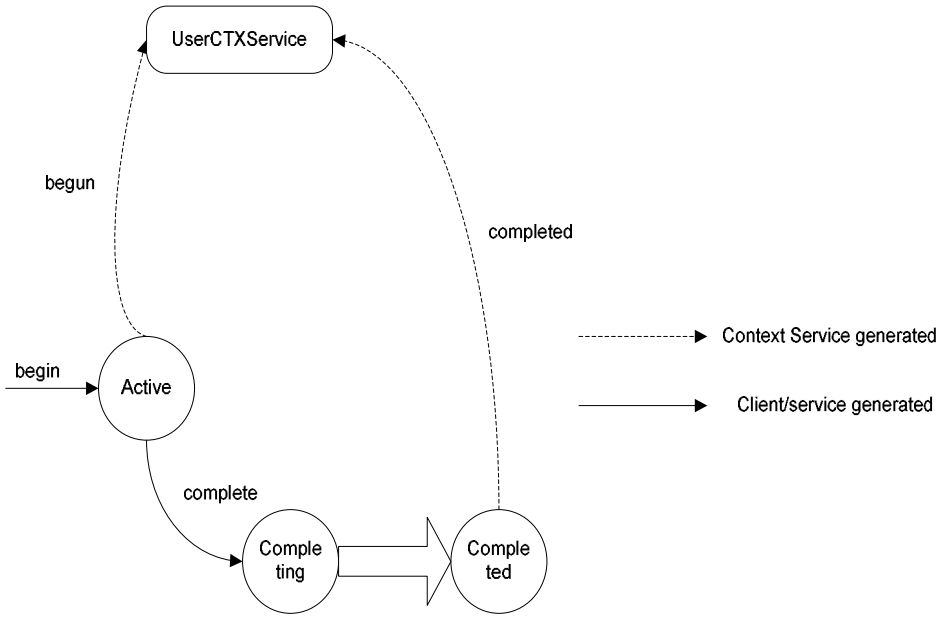


*Figure 12, Activity state transitions and messages.*

# 6 Security Considerations

WS-Context is designed to be composable with WS-Security. WS-Context provides a context structure that is typically bound to a SOAP header as well as endpoints for management of context lifecycle and contents.

It is RECOMMENDED that messages containing context headers use WS-Security[9] facilities for digital signatures to guarantee message integrity and to verify originators of both messages and contexts. The message as a whole, the individual context headers, or both may be signed. In addition, when contexts are passed by value sensitive context data should be encrypted with XML encryption facilities as described in WS-Security for confidentiality.

The ContextType schema includes an optional attribute, wsu:Id, which is used for ease of processing of WS-Security features. It is RECOMMENDED that implementations use the wsu:Id attribute to support encryption and signing of the context element. In addition, the context-identifier element definition includes an optional wsu:Id attribute to allow context services to sign identifiers, while allowing other services (e.g., the context manager) to freely update and change the content of the context itself.

It is RECOMMENDED that authorization checks be applied to context service and context manager operations. It is out of the scope of this specification to indicate how user identity and authorization are managed. Implementations may use appropriate mechanisms for the Web services environment. For example, user identity may be asserted via mechanisms described in Web Services Security Username Token Profile 1.0.

In addition to any authorization checks it may perform on the sender of a message, it is RECOMMENDED that applications services perform checks that contexts were created by authorized issuing authorities. A separate authorization problem arises for specific participation in specific activities. For example, a user may be permitted to access a service but not to participate in arbitrary transactions associated with the service. It is RECOMMENDED that application services maintain authorization checks for participation in specific activities based on domain specific requirements.

In order to defend against spoofing of context-identifiers by an attacker it is RECOMMENDED that service managers create context-identifiers incorporating random parts.

# 7 Conformance considerations

The WS-Context specification defines a session model for Web Services (the activity concept), a context to represent that model in executing systems and endpoints to manage context lifecycle and contents.

The minimum usage of WS-Context is restricted to the pass by value model of the context structure itself. Conformant implementations MUST follow the rules specified in Section 3; lexical representations of the context must be valid according to the schema definition for ctx:ContextType.

Systems and protocols that leverage the pass-by-reference representation of context MUST support the Context Manager. Conformant implementations of the Context Manager MUST follow the rules stated in Section 4.

Context lifecycle demarcation and control is managed by the Context Service. Conformant implementations of the Context Service MUST follow the rules stated in Section 5.

All messages based on the normative WSDL provided in this specification MUST be augmented by a Web services addressing specification to support callback-style message exchange.

Specifications that build on WS-Context MUST satisfy all requirements for referencing specifications that are identified for contexts, context-services and context managers.

# 8 References

[1] WSDL 1.1 Specification, see http://www.w3.org/TR/wsdl

[2] "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, S. Bradner, Harvard University, March 1997.

[3] "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, T. Berners-Lee, R. Fielding, L. Masinter, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.

[4] WS-Message Delivery Version 1.0, http://www.w3.org/Submission/2004/SUBM-ws-messagedelivery-20040426/

[5] WS-Reliability latest specification, http://www.oasis-open.org/committees/download.php/8909/WS-Reliability-2004-08-23.pdf. See Section 4.2.3.2 (and its subsection), 4.3.1 (and its subsections). Please note that WS-R defines BareURI as the default.

[6] Addressing wrapper schema, http://www.oasis-open.org/apps/org/workgroup/wsrm/download.php/8365/reference-1.1.xsd

[7] WS-R schema that uses the serviceRefType, http://www.oasis-open.org/apps/org/workgroup/wsrm/download.php/8477/ws-reliability-1.1.xsd

[8] Web Services Addressing, see http://www.w3.org/Submission/ws-addressing/

[9] Web Services Security: SOAP Message Security V1.0, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

# 9 Appendix A

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.