Web Services Context Specification (WS-Context) Version 1.0

OASIS Standard

2 April 2007

Specification URIs:

This Version:

http://docs.oasis-open.org/ws-caf/ws-context/v1.0/OS/wsctx.html

http://docs.oasis-open.org/ws-caf/ws-context/v1.0/OS/wsctx.pdf

http://docs.oasis-open.org/ws-caf/ws-context/v1.0/OS/wsctx.doc

Previous Version:

http://docs.oasis-open.org/ws-caf/ws-context/v1.0/CS01/wsctx.html

http://docs.oasis-open.org/ws-caf/ws-context/v1.0/CS01/wsctx.pdf

http://docs.oasis-open.org/ws-caf/ws-context/v1.0/CS01/wsctx.doc

Latest Version:

http://docs.oasis-open.org/ws-caf/ws-context/v1.0/wsctx.html

http://docs.oasis-open.org/ws-caf/ws-context/v1.0/wsctx.pdf

http://docs.oasis-open.org/ws-caf/ws-context/v1.0/wsctx.doc

Technical Committee:

OASIS Web Services Composite Application Framework (WS-CAF) TC

Chair(s):

Eric Newcomer (eric.newcomer@iona.com)

Martin Chapman (martin.chapman@oracle.com)

Mark Little (mark.little@jboss.com)

Editor(s):

Mark Little (mark.little@jboss.com)

Eric Newcomer (eric.newcomer@iona.com)

Greg Pavlik (greg.pavlik@oracle.com)

Related work:

This specification is related to:

- WS-Coordination Framework (part of OASIS WS-CAF)

- WS-Transaction Management (part of OASIS WS-CAF)

Declared XML Namespace(s):

http://docs.oasis-open.org/ws-caf/2005/10/wsctx

Status

This document was last revised or approved by the OASIS Web Services Composite Application Framework (WS-CAF) TC on the above date. The level of approval is also listed above. Check the current location noted above for possible later revisions of this document. This document is updated periodically on no particular schedule.

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

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

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

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 service definitions 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.

Notices

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.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to OASIS, except as needed for the purpose of developing OASIS specifications, in which case the procedures for copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it into languages other than English.

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

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

The names "OASIS", WS-Context and WS-CAF are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

Table of contents

1 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 http://example.org and http://example.com 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/ws-caf/2005/10/wsctx
```

1.1.1 Prefix Namespace

Prefix	Namespace
wsctx	**http://docs.oasis-open.org/ws-caf/2005/10/wsctx**
ref	**http://docs.oasis-open.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**
wsrm	**http://docs.oasis-open.org/wsrm/2004/06/reference-1.1.xsd**
soap	**http://schemas.xmlsoap.org/wsdl/soap/**
tns	**http://docs.oasis-open.org/ws-caf/2005/10/wsctx**

1.2 Referencing Specifications

One or more other specifications, such as (but not limited to) WS-Coordination Framework 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.

1.3 Precedence of schema and WSDL

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

2 Architecture

An activity represents the execution of a series of related interactions with a set of Web Services; these interactions 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. To achieve this, the fault mechanism of the underlying SOAP-based transport is used. For example, if an operation fails because no activity is present when one is required, then the callback interface will receive a SOAP fault including type of the fault and additional implementation specific information items supported the SOAP fault definition. WS-Context specific fault types are described for each operation. A fault type is communicated as an XML QName; the prefix consists of the WS-Context namespace and the local part is the fault name listed in the operation description.

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 1.1 binding is provided by default in this specification. A binding to WSDL 2.0 will be considered once that standard becomes more generally available and supported.

> 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.

> Note, conformant implementations MUST conform to the normative WSDL defined in the specification where those respective components are supported. Conformance with WSDL for optional components in the specification is REQUIRED only in the cases where the respective components are supported.

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 defined 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-Context 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: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, ServiceRefType.

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, if using WS-MessageDelivery specification [4] the value would be http://www.w3.org/2004/04/ws-messagedelivery. If using the WS-Addressing specification [8] then the value 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 reference to a Context Manager Service may appear as shown in Figure 3, where **ex** is an example namespace.

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

Figure 3, Web Service reference to a Context Manager service.

Figure 4 illustrates how an element derived from the **ServiceRefType** can be used as a container for a Web Service reference.

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

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.

> 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.

Referencing specifications extend the **wsctx:ContextType** both to identify the specific protocol type and extend the basic context structure to include protocol specific elements and attributes.

```
<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:element name="context-service" type="ref:ServiceRefType"
        minOccurs="0"/>
    <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="expiresAt" type="xsd:dateTime"
      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 **wsctx:contextIdentifierType** called **wsctx: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 **wsctx:contextIdentifierType** is a URI with an optional **wsu:Id** attribute. It MUST be unique.

- An OPTIONAL **wsctx:ServiceRefType** element, **wsctx:context-service**, which identifies the issuing authority responsible for generating the context.

- An OPTIONAL **wsctx:context-manager wsctx:ServiceRefType** to get data associated with a context-identifier that resolves to a reference to a Context Manager Web service. The presence of this endpoint is REQUIRED if the context has been passed by reference and it MAY be used to obtain the full value of the context later. It SHOULD NOT be present if the context is passed by value.

- An OPTIONAL **wsctx:parent-context** element containing some portion of the current context’s parent hierarchy.

- An OPTIONAL **wsctx:expiresAt** attribute, which indicates the date and time at which the context information expires; after this time, 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 **wsctx:expiresAt** attribute 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 **wsctx:expiresAt** 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 **wsctx: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.

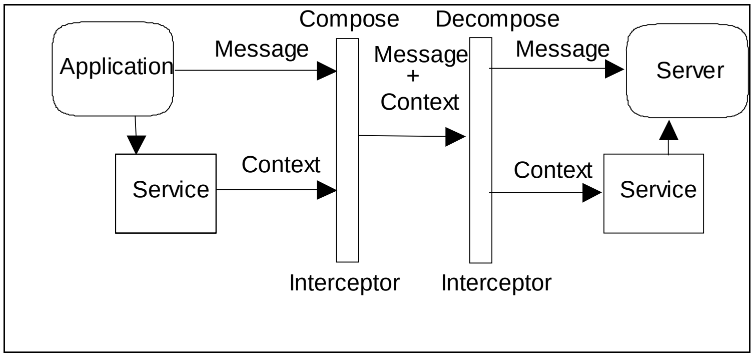

Figure 6, Services and context flow.

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 SHOULD return the entire current contents of the context, i.e. the values corresponding to the context's **wsctx:ContextType** elements held by the context service at the point of receiving the dereference message.

> Note, the ability of the context manager to return the context by value MAY be restricted by security considerations, e.g., if the invoker does not have the right privileges.

At a minimum, a context that is propagated by reference need only contain the **wsctx:context-identifier** and **wsctx:context-manager** elements. A context that is always propagated by value SHOULD NOT contain a **wsctx:context-manager** element. The endpoint should return a SOAP fault with the fault code set to the QName corresponding to **wsctx:InvalidContextStructure**.

> 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 in Section 2, 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 by the XML QName of the derived context type. 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. As a result, a SOAP header MAY contain multiple context data structures (**wsctx:ContextType**) representing different activities.

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 vary from insignificant or severe) and provide mechanisms that avoid operation execution in the context of 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 **wsctx:context-identifier**.

A referencing specification or implementation MAY use the **wsctx:InvalidContextStructure** fault code to indicate that a service has received a context structure that is invalid in a way defined by that referencing specification.

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 QName.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2002/06/soap-envelope">
  <soap:Header>
    <example:context
        xmlns="http://docs.oasis-open.org/ws-caf/2005/10/wsctx"
        expiresAt="2005-04-26T22:50:00+01:00"
        xmlns:wsdl=”http://schemas.xmlsoap.org/wsdl/”
        xmlns:soapbind=”http://schemas.xmlsoap.org/wsdl/soap/”
        xmlns:example=”http://example.com/context/”
        soap:mustUnderstand="1">
      <context-identifier>
        http://docs.oasis-open.org/ws-caf/2005/10/wsctx/abcdef:012345
      </context-identifier>
      <context-service>
        <example:address>
          http://example.org/wsctx/service
        </example:address>
      </context-service>
         <parent-context expiresAt="2005-04-27T22:50:00+01:00">
        <context-identifier>
          http://example.org/5e4f2218b
        </context-identifier>
        <context-service>
          <example:address>
            http://example.org/wsctx/service
          </example:address>
           </context-service>
      </parent-context>
    </example:context>
  </soap:Header>
  <soap:Body>
    <!-- Application Payload -->
  </soap:Body>
</soap:Envelope>
```

Figure 7, Context Transported in a SOAP Header Block.

4 Context Manager

As described in Section 3, a context MAY be passed by reference or by value. If the context is passed by reference, then a receiver may eventually require the context's value information. WS-Context defines 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 (see Section 5).

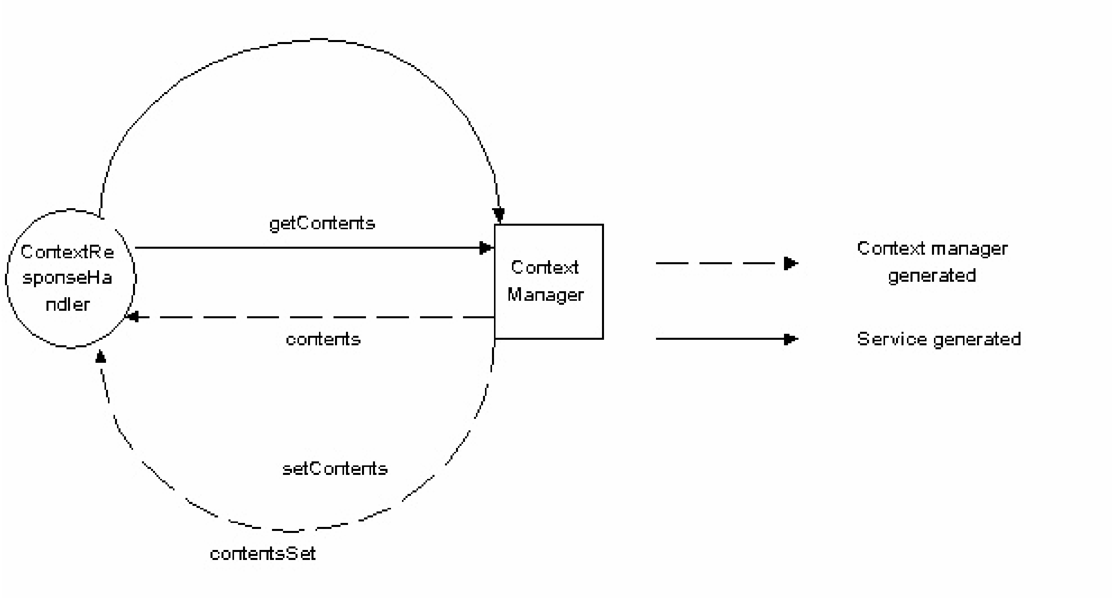

Figure 8, Context interactions.

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. The Context Manager responds with either the *contents* message or an appropriate fault message. The entire contents of the context SHOULD be returned, i.e. the values corresponding to the context's ContextType elements. Note, the implementation MAY impose restrictions based on security privileges, for example.
- *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, 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.
- *UnknownContext*: this fault code is sent to indicate that the specified context cannot be located.

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: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 its existence an activity MAY report statuses (which SHOULD unambiguously reflect internal states of the activity), in reaction to receipt of the message **wsctx:getStatus**.

The referencing specification states whether statuses will be reported, and if so, how possible states are named and defined. If an activity does not return statuses then it MUST return a fault **wsctx:NoStatusesDefined** when asked to report a status.

If a Context Service does return statuses then it MUST report its current status when asked; there is no notion of automatically informing services when a specific state is entered. If an activity cannot report its current status but may be able to do so in the future then it SHOULD return a fault **wsctx:StatusUnknown**. If an activity is unknown to the Context Service when it is asked to report a status, then it SHOULD return a fault: **wsctx:UnknownActivity**.

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 MUST 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 UserContextService and ContextService interfaces respectively. The WSDL for the PortTypes of these services is shown below and the interactions are described in the following section.

```
<wsdl:portType name="ContextServicePortType">
  <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="UserContextServicePortType">
  <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="timeoutSet">
     <wsdl:input message="tns:TimeoutSetMessage"/>
   </wsdl:operation>
   <wsdl:operation name="timeout">
     <wsdl:input message="tns:TimeoutMessage"/>
   </wsdl:operation>

  </wsdl:portType>
```

Figure 10, ContextService WSDL.

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

- ContextService: this has operations begin, complete, getStatus, setTimeout and getTimeout;
- UserContextService: this is the user/service callback endpoint address for the various ContextService operations. As such, it has operations begun, completed, status, timeoutSet, timeout.

The ContextService 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 **wsctx: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 **wsctx:InvalidContext** fault will be sent to the received UserContextService endpoint.

If nesting of activities is not supported by the implementation and there is a context present with the *begin* message then **wsctx:InvalidContextStructure** fault will be sent to the UserContextService endpoint.

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

- *any dateTime value*: the Activity MUST complete by the expiry date and time.

- not present: the Activity will never be completed automatically by the Context Service implementation, i.e., it will never be considered to have timed out. If the implementation does not support this semantic, then the **wsctx:TimeoutNotSupported** fault will be sent to the UserContextService.

- *empty*: the last value specified using *setTimeout* is used. If no prior call to the *setTimeout* operation has occurred for this thread, or the duration 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 the **wsctx:TimeoutNotSupported** fault being sent to the UserContextService endpoint.

Upon success, the *begun* response will be sent by invoking the begun operation of the UserContextService. The context will be present as a SOAP header in envelope containing the begun message.

If an invalid context is propagated on the begin request then the **wsctx:InvalidContext** fault code is returned to the UserContextService.

The **wsctx:InvalidProtocol** fault is sent to the UserContextService 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 **wsctx:NoPermission** fault MAY be returned to the UserContextService. 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 UserContextService. If the activity is in a state where completed is not allowed (eg, the activity has already completed), then the **wsctx:InvalidState** fault will be sent to the UserContextService.

If an invalid context is propagated on the request then the **wsctx:InvalidContext** fault is sent to the UserContextService.

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 UserContextService to return the current status of the Activity. If there is no valid context associated with the context-identifier, the **wsctx:InvalidContext** fault code is returned to the UserContextService.

If an invalid context is propagated on the request then the **wsctx:InvalidContext** fault code is returned to the UserContextService.

setTimeout

No context is associated with this invocation. This operation modifies a state variable associated with the Context Service that affects the expiry date and time associated with the activities created by subsequent invocations of the begin operation when no expiry is specified (i.e., the begin expiresAt value is empty): 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 duration*: the Activity MUST complete within this duration from the time the activity is begun.
- *Not present*: the Activity will never be completed automatically by the Context Service implementation, i.e., it will never be considered to have timed out. If the implementation does not support this semantic, then the **wsctx:TimeoutNotSupported** fault code will be sent to the UserContextService.
- *0*: it is implementation dependant as to the meaning of passing a zero duration.

A valid timeout value results in the Context Service calling the UserContextService's *timeoutSet* operation. Any other value results in the **wsctx:TimeoutNotSupported** fault code being invoked on the associated UserContextService.

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 duration that is associated with activities created by calls to begin when no expiresAt value is passed via begin.

5.2.1 WS-Context Faults

This section defines well-known error codes to be used in conjunction with an underlying fault handling mechanism.

Unknown Context

This fault is sent by the ContextManager to indicate that the context identified in a received message is not recognised. This may indicate an unknown activity.

The qualified name of the fault code is:

```
wsctx:UnknownContext
```

Invalid Context

This fault can be sent by an endpoint to indicate that it cannot accept a context which it was passed.

The qualified name of the fault code is:

```
wsctx:InvalidContext
```

No Context

This fault can be sent by an endpoint to indicate that it did not receive a context when one was expected.

The qualified name of the fault code is:

```
wsctx:NoContext
```

Invalid State

This fault is sent by the Context Service to indicate that the endpoint that generates the fault has entered an invalid state. This is an unrecoverable condition.

The qualified name of the fault code is:

```
wsctx:InvalidState
```

Invalid Context Structure

This fault it sent by the Context Service if nesting of activities is not supported and there is a context present with the *begin*. This is an unrecoverable condition.

The qualified name of the fault code is:

```
wsctx:InvalidContextStructure
```

Timeout Not Supported

This fault is sent by the Context Service if an attempt is made to create an activity without a timeout and the implementation does not support that semantic. This is an unrecoverable condition.

The qualified name of the fault code is:

```
wsctx:TimeoutNotSupported
```

Parent Activity Completed

This fault is sent by the Context Service if an attempt is made to create a nested activity with a parent activity that has already completed. This is an unrecoverable condition.

The qualified name of the fault code is:

```
wsctx:ParentActivityCompleted
```

No Permission

This fault MAY be sent by the Context Service if the implementation imposes restrictions on which Web services can terminate an activity.

The qualified name of the fault code is:

```
wsctx:NoPermission
```

Child Activity Pending

This fault MAY be sent by the Context Service if an attempt is made to complete a parent activity that currently has active child activities.

The qualified name of the fault code is:

```
wsctx:ChildActivityPending
```

Status Unknown

This fault SHOULD be sent by a Context Service if it cannot report its current status but may be able to do so in the future.

The qualified name of the fault code is:

```
wsctx:StatusUnknown
```

No Statuses Defined

This fault MUST be sent by a Context Service if a status value is requested and no values have been defined by the referencing specification.

The qualified name of the fault code is:

```
wsctx:NoStatusesDefined
```

Unknown Activity

This fault SHOULD be returned if an activity is unknown to the Context Service when it is asked to report a status.

The qualified name of the fault code is:

```
wsctx:UnknownActivity
```

Invalid Protocol

This fault is be sent by the Context Service if an attempt is made to create an activity with a protocol type it does not recognise.

The qualified name of the fault code is:

```
wsctx:InvalidProtocol
```

5.2.2 Message exchanges

The WS-CAF protocol family is defined in WSDL, with associated schemas. All the WSDL has a common pattern of defining paired port-types, such that one port-type is effectively the requestor, the other the responder for some set of request-response operations.

portType for an initiator (“client” for the operation pair) will expose the responses of the “request/response” as input operations (and should expose the requests as output messages); the responder (service-side) only exposes the request operations as input operations (and should expose the responses as output messages).

Each “response” is shown on the same line as the “request” that invokes it. Where there are a number of responses to a “request”, these are shown on successive lines. The initiator portTypes typically include various fault and error operations.

Initiator (and receiver of response)	**Responder**	**“requests”**	**responses**

Initiator (and receiver of response)	**Responder**	**“requests”**	**responses**
ContextResponseHandler	**ContextManager**	setContents	contentsSet wsctx:UnknownContext wsctx:InvalidContext wsctx:NoContext
		getContents	contents wsctx:UnkownContext wsctx:InvalidContext wsctx:NoContext
UserContextService	**ContextService**	begin	begun wsctx:InvalidState wsctx:InvalidContext wsctx:InvalidContextStructure wsctx:TimeoutNotSupported wsctx:ParentActivityCompleted wsctx:NoPermission wsctx:InvalidProtocol
		complete	completed wsctx:InvalidState wsctx:InvalidContext wsctx:ChildActivityPending wsctx:NoPermission wsctx:NoContext
		getStatus	status wsctx:InvalidState wsctx:InvalidContext wsctx:NoPermission wsctx:NoContext
		setTimeout	timeoutSet wsctx:InvalidState wsctx:InvalidContext wsctx:TimeoutNotSupported wsctx:NoPermission
		getTimeout	timeout wsctx:InvalidState wsctx:InvalidContext wsctx:NoPermission

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 block 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 **wsctx: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 Normative 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

Appendix A. Acknowledgements

The following individuals were active members of the committee during the development of this specification:

Kevin Conner (Arjuna Technologies)
Mark Little (Arjuna Technologies)
Tony Fletcher (Choreology)
Peter Furniss (Choreology)
Alastair Green (Choreology)
John Fuller (Individual)
Eric Newcomer (IONA Technologies)
Martin Chapman (Oracle)
Simeon Green (Oracle)
Jeff Mischinkinsy (Oracle)
Greg Pavlik (Oracle)
Pete Wenzel (SeeBeyond)
Doug Bunting (Sun Microsystems)

Thanks to all members, past and present, of the WS-CAF technical committee who contributed to the various versions of the specification.