**Issue 103** – Standardizing $varName syntax for XPath to refer to a BPEL variable

103 Proposal Base For Delta: Feb 14, 2005 – 6 pm PST

Last modified: May 19, 2005 – 7pm PDT

**Proposal**: Map BPEL variables directly to XPATH 1.0 variables (e.g. $foo) and so get rid of getVariableData. Also provide full clarification for how BPEL and XPATH 1.0 work together.

**Rationale**: The spec currently is very light on the specific details of how exactly BPEL data works with XPATH 1.0. For example, how XML Schema data types map to XPATH 1.0 data types or what the value of the context node is in XPATHs used in query expressions. The spec also requires the use of the getVariableData XPATH 1.0 function to access variable data from within XPATH 1.0. But XPATH 1.0 provides a native facility for variable access that provides a more natural experience for XPATH programmers, provides more readable code and prevents some very annoying edge cases possible with getVariableData (see issue 29).

**Changes Required**: Remove the part attribute from propertyAlias element. Re-arrange section 9 and add in a new sub-section (9.1.1) on how to manifest WSDL message as infosets (each part in the WSDL message is a stand alone infoset whose document element has the part's name) and another section to define XPATH query and expression language behavior.

# Section 8.2

[***Editor's Note: AYIU: Issue 203: Abandon this part of "from-to" changes about propertyAlias : ###BEGIN]

**From:**
The syntax for a propertyAlias definition is:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         1         2         3         4         5         6
<definitions name="ncname"
        ... >
        <bpws:propertyAlias propertyName="qname"
               messageType="qname"? part="ncname"?
               type="qname"? element="ncname"?>
           <query queryLanguage="anyURI"?>?
           ... queryString ...
           </query>
        </bpws:propertyAlias>
        ...
```

```
</wsdl:definitions>
```

A property alias element MUST use one of the three following combinations of attributes:

- messageType and part,
- type or
- element.

When a propertyAlias is used with the messageType/part combination the messageType specifies that all BPEL variables defined with the same messageType MUST manifest the property. The part attribute and query element and then applied against the BPEL messageType variable to either set or get the property variable in the same way that the part attribute and query element are used in the first from and to specs in copy assignments.

**To:**
The syntax for a propertyAlias definition is:

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         1         2         3         4         5         6
<definitions name="ncname"
        ... >
        <bpws:propertyAlias propertyName="qname"
               messageType="qname"?
               type="qname"? element="ncname"?>
           <query queryLanguage="anyURI"?>?
           ... queryString ...
           </query>
        </bpws:propertyAlias>
        ...
</wsdl:definitions>
```

A property alias element MUST use exactly one of the following attributes:

- messageType,
- type or
- element.

The getVariableProperty function MUST manifest a property for any variable for which there exists a propertyAlias definition that associates the property with the same declaration (e.g. messageType, type or element) as the one used to define the variable. The value of the property will be generated by applying the query string inside the <query> element to the variable's value. The query string will be processed in the same manner as a corresponding to-spec/from-spec in copy assignments.

[***Editor's Note: AYIU: Issue 203: Abandon this part of "from-to" changes about propertyAlias: ###END]

[***Editor's Note: AYIU: However, we can still add the following paragraph after the paragraph started with "When a propertyAlias is used with the messageType/part

combination … ” with some minor syntax touch-up.]

Using the same “`tns:taxpayerNumber`” example from above, for a message variable “`myTaxPayerInfo`” of type “`txmsg:taxpayerInfo`”:

```
<from variable="myTaxPayerInfo" property="tns:taxpayerNumber" />
```

and

```
<from>$myTaxPayerInfo./identification/txtyp:socialsecnumber</from>
```

have the same output. Please see **Assignment** for details.

[***Editor’s Note: AYIU: making simple syntax changes above from “/” to “.”]

# Changes in Section "9. Data Handling"

Section 9.2 "Variables" would become Section 9.1. Section 9.1 "Expressions" would become Section 9.3 and a new section 9.2 "Usage of Query and Expression Languages" would be inserted. Section 9.3 "Assignment" would then become section 9.4. In other words:
9.1 Variables (formerly 9.2)
9.2 Usage of Query and Expression Languages (Brand New)
9.3 Expressions (formerly 9.1)
9.4 Assignment (formerly 9.3)

# Old Section 9.2/Now Section 9.1

After the sentence of “The name of a variable should be unique within its own scope”, add the following sentences:

Variable names are NCNames (as defined in XML Schema specification) but in addition they MUST NOT contain the “.” character. This restriction is necessary because the “.” character is used as a delimiter in variable names in BPEL's default binding to XPath 1.0 (i.e. the binding identified by "urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"). The delimiter separates the WS-BPEL message type variable name and the name of one of its WSDL message parts. The concatenation of the WSDL message variable name, the delimiter and the WSDL part name is used as an XPath variable reference which manifests the XML Infoset of the corresponding WSDL message part.

Pre-pend the following before the paragraph (added by issue 155) that begins “The infoset for a complexType variable consists of a document information item…”: Using

[Infoset] terminology, the infoset for a BPEL element variable consists of a document information item (DII) that contains exactly one child, an element information item (EII) which is pointed at by the document element property. The EII is the value of the element variable. If a BPEL implementation chooses to manifest a simpleType variable as an XML infoset**,** the infoset should consist of a DII that contains exactly one child, an EII which is pointed at by the document element property. The properties of the document element, specifically the namespace name and local name properties, are undefined by this specification, as such an implementation MUST specify whatever namespace name/local name values it likes. However the children of the document element MUST exclusively consist of a series of character information items (CharIIs) that represent the simpleType value.A BPEL implementation MAY choose to map simpleType variables to non-XML-infoset data-models defined in the expression/query language being used. (e.g. Boolean in XPath 1.0)

In order to simplify data access, WSDL parts of WSDL message variables are manifested in BPEL as ~~a collection of~~ infosets, one infoset per WSDL message part. BPEL engines MUST follow the following algorithm when manifesting a WSDL message part~~s~~ as an infoset~~s~~.

For each part in the WSDL message definition:
Step 1 – Create a synthetic DII which has no children other than as specified below.
~~Step 2 - Create an EII as a child of the document element whose namespace name is null and whose local name is the same as the WSDL message part's name.~~
Step ~~3~~2a - If the WSDL message part is defined using the type attribute then create an EII as a child of the document element. The local name and namespace name of the newly created EII are determined by the WS-BPEL 2.0 engine and are not specified by this document. The handling of this EII is similar to how WS-BPEL 2.0 handles the containers for complex and simple type XML variables. The contents of the new EII are required to conform to the contents defined by the referenced type definition.
Step ~~3b~~2b - If the WSDL message part is defined using the element attribute then ~~the new EII will itself contain~~ create an EII as a child of the document element which manifests the element defined by the referenced type definition.

The previous models are conceptual; they define how WS-BPEL 2.0 submits and retrieves XML variable values using infoset definitions. But these models are not intended to require that WS-BPEL 2.0 implementations actually implement an infoset model, only that however variable binding is handled the end result duplicates the behaviors defined using the infoset model. For example, a WS-BPEL 2.0 implementation may choose to bind a simple type BPEL variable of type xs:string directly to a String object in XPath 1.0. The choice of mapping MUST be consistently applied to variables and WSDL message part values of the same XML schema type. E.g. if a xs:string variable is manifested as a string object, a xs:string message part MUST be manifested as a string object also. For detailed definition of manifestation of BPEL variables in XPath 1.0, please see “Binding BPEL Variables In XPath 1.0” section.

In summary, a BPEL variable is manifested as XML Infoset items in one of the following ways:

(1) a single XML infoset item: e.g. an element or complexType variable or a WSDL message part

(2) ~~a collection of XML infoset items: e.g. When XPath 1.0 is used, a WSDL message variable is manifested as a nodeset~~

(3) a sequence of Character Information Items for simple type data: e.g. used to manifest string (Please note these items may be manifested as a non XML infoset item when needed. e.g. Boolean)

# 9.2 Usage of Query and Expression Languages

This section models the interaction between Query/Expression languages and the BPEL process from two different perspectives. The first perspective is BPEL's view of the query/expression languages. That view is restricted to what information BPEL will make available for use by the Query/Expression language. The second perspective is the Query/Expression language's view of BPEL, specifically XPath 1.0 and how XPath 1.0's execution context is initialized by BPEL.

## 9.2.1 Enclosing Elements

In order to describe the view that BPEL provides to Query/Expression languages it is necessary to introduce a new term - Enclosing Element.

Definition of an Enclosing Element of a Query or Expression language: Whenever a query or expression language is used in BPEL (e.g. XPath 1.0), the Enclosing Element is defined as the parent element in the BPEL process definition that contains the Query or Expression. For example, in the following from specification example:

```
<process>
   ...
   <from>$myVar/abc/def</from>
   ...
</process>
```

The "from" element is the Enclosing Element.

The in-scope namespaces of the enclosing element are the namespaces visible to the Query/Expression language. (Note: XPath 1.0 does not have default namespace concept.)

The links, variables, partnerLinks, fault handlers, compensation handlers, etc. that are visible to a Query/Expression language are defined based on the visibility of those entities to the activity that the enclosing element is contained within. There is no requirement that

a Query/Expression language must manifest all the different objects previously described, only that if such objects are accessible within the Query/Expression language then only the objects in scope to the enclosing element's enclosing activity SHOULD be visible from within the Query/Expression language.

Evaluation of a BPEL expression or query will yield one of the following: (here we use XPath 1.0 expressions as examples)

(1) a single XML infoset item: e.g. `$myFooVar/lines/line[2]`
(2) a collection of XML infoset items e.g. `$myFooVar/lines`
(3) a sequence of Character Information Items for simple type data
e.g. `$myFooVar/lines/line[2]/text()`
(Please note this sequence of items may be manifested as a non XML infoset item when needed. e.g. boolean)
(4) a variable reference: e.g. `<from>$myFooVar</from> <to>$myBarVar</to>`

### 9.2.2 Binding BPEL Variables In XPath 1.0

With the exception of Link expressions whose variable access syntax and semantics are described in section 9.2.7, BPEL variables are accessible in XPath expressions in BPEL processes via XPath 1.0 variable bindings. Specifically, all BPEL variables visible from the enclosing element of an XPath 1.0 expression MUST be made available to the XPath 1.0 processor by manifesting the BPEL variable as an XPath variable binding whose name is the same as the BPEL variable's name, except the case of WSDL message variables declared with a messageType, which requires some special handling (discussed below).

BPEL variables declared using an element are manifested as a node-set XPath variable with a single member node. That node is a synthetic DII that contains a single child, the document element, which is the value of the BPEL variable. The XPath variable binding will bind to the document element. For example, given the following schema definition:

```
<xsd:element name="StatusContainer">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="statusDescription" type="xs:string" form="qualified" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

And given the following variable declaration:

```
<variable name="AStatus" element="e:StatusContainer"/>
```

Then a BPEL XPath expression to access the value of the statusDescription element, assuming the AStatus variable is in scope, would look like:

$Astatus/e:statusDescription

$AStatus points at the variable's document element, StatusContainer. So to access StatusMessage's child statusDescription it is only necessary to specify the child's element name.

BPEL variables declared using a complex type are manifested as a node-set XPath variable with one member node that contains the anonymous document element that itself contains the actual value of the BPEL complex type variable. The XPath variable binding will bind to the document element. For example, given the following schema definition:

```
<xs:complexType name="AuctionResults">
  <xs:sequence>
    <xs:element name="AuctionResult" maxOccurs="unbounded" form="qualified">
      <xs:complexType>
        <xs:attribute name="AuctionID" type="xs:int"/>
        <xs:attribute name="Result" type="xs:string"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

And given the following variable declaration:

<variable name=”AuctionResults” type=”e:AuctionResults”>

Then a BPEL XPath expression to access the value of the second AuctionID attribute would look like:

$AuctionResults/e:AuctionResult[2]/@AuctionID

$AunctionResults points at the variable’s document element, AuctionResult[2] points to the second AuctionResult child of the document element, and @AuctionID points to the AuctionID attribute on the selected AuctionResult element.

WS-BPEL 2.0 messageType variables are manifested in XPath 1.0 as a series of variables, one variable per part in the messageType. Each variable is named by concatenating the message variable's name, the "." character and the local name of the part. The data in a BPEL messageType variable is NOT made available as one single XPath variable to general XPath processing under the default query and expression language binding. For example, if a messageType variable was named "myMessageTypeVar" and it contained two parts, "msgPart1" and "msgPart2" then XPath 1.0 binding that had "myMessageTypeVar" in scope would manifest two XPath 1.0 variables, $myMessageTypeVar.msgPart1 and $myMessageTypeVar.msgPart2. Please

note that section 2.3 of [WSDL 1.1] requires that all part names within the same WSDL message definition must be unique.

WSDL message parts are always defined using either an XSD element, an XSD complex type or a XSD simple type. As such the manifestation of these message parts in XPath can be handled in the same manner as specified herein for element, complex type and simple type WS-BPEL 2.0 variables.

Below is a full example of how a WSDL message type is manifested in WS-BPEL 2.0 XPath.

~~BPEL variables declared using a messageType are manifested as a node-set XPath variable with one member node for each part. The XPath variable binding will bind to the document element. Because the parts are contained in a node-set and because node-sets in XPath are unordered there is a potential problem, it is impossible using node-sets to distinguish between two nodes in the node-set with the same type and name. Thankfully that is not an issue with WSDL messages as section 2.3 of [WSDL 1.1] requires that all part names within the same WSDL message definition must be unique. For example, given the following WSDL message definition:~~

```
<message name="StatusMessage">
  <part name="StatusPart1" element="e:StatusContainer"/>
  <part name="StatusPart2" element="e:StatusContainer"/>
</message>
```

And given the following variable declaration:

<variable name=”StatusVariable” messageType=”e:StatusMessage”>

Then a BPEL XPath expression to access the second part’s statusDescription element would look like:

$StatusVariable~~./~~StatusPart2/~~e:StatusContainer/~~e:statusDescription

[***Editor’s Note: AYIU: making simple syntax changes above from “/” to “.” and droppoing “e:StatusContainer”]

**Note:** It is possible to write XPath 1.0 queries that can simultaneously query across multiple parts of a WSDL message variable by applying a union operator to create one single nodeset. For example:

( $StatusVariable.StatusPart1 | $StatusVariable.StatusPart2 )//e:amount

BPEL simpleType variables are manifested directly as either an XPath string, Boolean or float object. If the XML schema type of the BPEL simpleType variable is xs:Boolean or any types that are restrictions of xs:Boolean then the BPEL variable will be manifested as

an XPath Boolean object. If the XML schema type of the BPEL simpleType variable is xs:float, xs:int, xs:unsignedInt or any restrictions of those types then the BPEL variable will be manifested as an XPath float object. Any other XML schema types will be manifested as an XPath string object.

XPath 1.0 only has a single numeric object, float. Float is a 64 bit IEEE floating point number. However the resolution of the float object is not sufficient to capture the full value of some XML Schema data types. For example, a XSD decimal number must support at least 18 digits where as an XPath float only has to support approximately 16 digits. XSD numeric values that cannot be expressed without loss of accuracy as XPath float objects are instead translated into XPath string objects. XPath string objects can be explicitly translated into XPath float objects using the XPath number() function. Similarly if an XPath string object is used in a situation that calls for an XPath float object the XPath processor will automatically translate the XPath string object to an XPath float object. But implementers should be aware that if the accuracy of the XML schema type is greater than supported by the XPath float object then accuracy will be lost during the translation from string object to float object.

### 9.2.3 XPath 1.0 perspective and BPEL

Section 1 of the XPath 1.0 specification [XPATH 1.0] defines five points that define the context in which an XPath expression is evaluated. Those points are reproduced below:

- a node (the context node)
- a pair of non-zero positive integers (the context position and the context size)
- a set of variable bindings
- a function library
- the set of namespace declarations in scope for the expression

The following sections define how these contexts are initialized in BPEL for different types of BPEL expression and query language contexts.

#### 9.2.3.1 Default use of XPath 1.0 for Expression and Query Languages

When XPath 1.0 is used for a Query or Expression Language, except as otherwise specified, the XPath context is initialized as follows:

- Context node = none
- Context position = none ; Context Size = none
- A set of variable bindings = variables visible to the Enclosing Element as defined by the BPEL scope rules
- A function library = core XPath and BPWS function libraries MUST be available and an engine specific function library MAY be available
- Namespace declaration = in-scope namespace declarations from Enclosing Element

It is worth emphasizing that as defined by the XPath 1.0 standard when resolving an XPath the namespace prefixes used inside of the variable (e.g. BPEL variables) are irrelevant. The only prefixes that matter are the in-scope namespaces.

For example, imagine a BPEL variable named “FooVar” of “foo” element type with value:
<a:foo xmlns:a="http://example.com"><a:bar >23</a:bar></a:foo>

The following XPath would return the value 23:
<from xmlns:b="http://example.com">$FooVar/b:bar/text()</from>

Notice that in the previous example the bar element is referred to use the 'b' namespace prefix rather than the 'a' namespace prefix that is used inside the actual value.

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

Using the same value for Foo as provided previously the following would return a bpws:selectionFailure fault (in executable BPEL), because it fails to select any node:
<bpws:from xmlns="http://example.com">$FooVar/bar/text()</bpws:from>

The values inside of the XPath do not inherit the default namespace of the enclosing element. So the 'bar' element referenced in the XPath does not have any namespace value what so ever and therefore does not match with the bar element in the FooVar variable which has a namespace value of http://example.com.

Allowing BPEL variables to manifest as XPath variable bindings enables BPEL programmers to create powerful XPath expressions involving multiple BPEL variables. For example:

```
<assign>
   <copy>
      <from>$po/lineItem[@prodCode=$myProd]/amt * $exchangeRate</from>
      <to>$convertedPO/lineItem[@prodCode=$myProd]/amt</to>
   </copy>
</assign>
```

When XPath 1.0 is used as an expression or query language in BPEL, with the exception of propertyAlias definitions, there is no context node available. Therefore the legal values of the XPath Expr (http://www.w3.org/TR/xpath#NT-Expr) production must be restricted in order to prevent access to the context node.

Specifically, the "LocationPath" (http://www.w3.org/TR/xpath#NT-LocationPath) production rule of "PathExpr" (http://www.w3.org/TR/xpath#NT-PathExpr) production rule MUST NOT be used when XPath is used as an expression or query language (except in the case of propertyAlias which is covered separately). The previous restrictions on the

XPath Expr production for the use of XPath as an expression language MUST be statically enforced.

The result of this restriction is that the "PathExpr" will always start with a "PrimaryExpr" (http://www.w3.org/TR/xpath#NT-PrimaryExpr) for BPEL expression or query language XPaths. It is worth remembering that PrimaryExprs are either variable references, expressions, literals, numbers or function calls, none of which can access the context node.

A query language is used when an lvalue needs to be returned. In addition to the previously listed restrictions, which except where otherwise indicated MUST apply to XPaths used as a query language, additional restrictions are also needed to ensure that the returned value of a query language expression is an lvalue. Specifically, when XPath is used as a query language the XPath MUST begin with a "VariableReference" and this restriction MUST be statically enforced.

### 9.2.3.2 Use of XPath 1.0 for Expression Languages in Join Conditions

When XPath 1.0 is used as an Expression Language for a join condition, the XPath context is initialized as follows:

- Context node = none
- Context position = none ; Context Size = none
- A set of variable bindings = links that the target the activity that the Enclosing Element is contained within
- A function library = the core XPath function library MUST be available, the BPWS function library MUST NOT be available and an engine specific function library MAY be available.
- Namespace declaration = in-scope namespace declarations from Enclosing Element

As explained in section 12.5.1 expressions in join conditions may only access the status of links that target the join condition's enclosing activity. No other data may be made available. To this end the only variable bindings made available to join conditions are ones that access link status. Also note that link status is only available in join conditions which is why links are not bound to XPath variables in any other context.

Link status is made available via XPath variable bindings by manifesting links that target the activity that contains the Enclosing Element as XPath variable bindings of identical name. That is, if there is a link called "ABC" that targets the activity then there must be an XPath variable binding called "ABC". Link variables are manifested as XPath Boolean objects whose value will be set to the Link's value.

Below is an example of a joinCondition inside of a <targets> element:

```
<targets>
   <target linkName="link1"/>
   <target linkName="link2"/>
   <joinCondition>$link1 and $link2</joinCondition>
```

```
</targets>
```

### 9.2.3.3 Use of XPath 1.0 for Query Languages in propertyAliases

[***Editor's Note: AYIU: Issue 203 Changes … the below extra changes are resulted from Issue 145.]

When XPath 1.0 is used as Query Language for a propertyAlias, the XPath context is initialized as follows:

- Context node = ~~a node-set that contains a node (an EII) for each part in the WSDL message definition~~
  - When messageType/part attributes are used:
    - If the message part is based on a complex type or an element, the context node will point to node-list containing a single node which is the EII for the referenced part where the EII is defined as specified in section 9.2.2.
    - If the message part is based on a simple type, the context node will point to the XPath object specified for the particular variable type in section 9.2.2.
  - When type attribute is used:
    - If the type is a complex type, the context node will point to node-list containing a single node which is the EII for the referenced part where the EII is defined as specified in section 9.2.2.
    - If the type is a simple type, the context node will point to the XPath object specified for the particular variable type in section 9.2.2.
  - When element attribute is used, the context node will point to node-list containing a single node which is the EII for the referenced part where the EII is defined as specified in section 9.2.2.
- Context position = 1 ; Context Size = 1
- A set of variable bindings = There MUST NOT be any variable bindings available when XPath is used as query language in propertyAlias
- A function library = The core XPath function library MUST be available, the BPWS function library MUST NOT be available and an engine specific function library MAY be available.
- Namespace declaration = in-scope namespace declarations from Enclosing Element (note that in the past enclosing elements were always themselves enclosed within an activity, that is not the case here since propertyAliases are defined in WSDL files. But since there are no variable bindings to worry about this difference should not matter.)

propertyAlias is unique in BPEL in that it has a defined context node. Therefore none of the previously listed restrictions on the syntax of the XPath expression apply here. Any legal XPath expression may be used. It does not matter if an absolute or relative path is used in a propertyAlias as both will resolve to the context node since the context node in this case is the root node.

For example:
<propertyAlias propertyName="poId" messageType="my:POMsg" part="poPart">
<query> ~~poPart/~~po/id</query>
</propertyAlias>

Or

<propertyAlias propertyName="poId" messageType="my:POMsg" part="poPart">
<query>(~~poPart/~~po/id + 1)</query>
</propertyAlias>

There is no requirement that property aliases return lvalues but it is worth pointing out that an attempt to assign to a property alias that doesn't specify a lvalue will, as defined in section 9.3, fail with a bpws:selectionFailure.

### 9.2.8 BPEL XPath Functions

See section below entitled 'Changes in Section "14.1. Expressions"'.

# Old Section 9.1/Now Section 9.3

Delete the paragraphs:

```
bpws:getLinkStatus ('linkName')
```

This function returns a Boolean indicating the status of the link (see Link Semantics). If the status of the link is positive the value is true, and if the status is negative the value is false. This function MUST NOT be used anywhere except in a join condition. The linkName argument MUST refer to the name of an incoming link for the activity associated with the join condition. These restrictions MUST be statically enforced.

# Old Section 9.3/Now Section 9.4

The From-Spec list given in the text is to be changed **To:**
<from variable="ncname" part="ncname"? />
~~<from expressionLanguage="anyURI"?>expression</from>~~
<from partnerLink="ncname" endpointReference="myRole|partnerRole"/>
<from variable="ncname" property="qname"/>
<from expressionLanguage="anyURI"?>expression</from>
<from> <literal~~="yes"~~> ... literal value ... </literal> </from>

-
The To-Spec list given in the text is to be changed **To:**
<to variable="ncname" part="ncname"? />

~~<to queryLanguage="anyURI"?>$varRef/locationPath</to>~~
<to partnerLink="ncname"/>
<to variable="ncname" property="qname"/>

<to queryLanguage="anyURI"?>… query … </to>

[***Editor's Note: AYIU: We should make variable-to-variable copy available again by adding the first form of from-spec and to-spec, because we don't manifest BPEL message variable as a single XPath variable anymore. Adding back the first form allows us to continue doing msg part copy without using any expression language. ]

**Keep ~~From~~:** In the first from-spec and to-spec variants the variable attribute provides the name of a variable. If the type of the variable is a WSDL message type the optional part attribute MAY be used to provide the name of a part within that variable. When the variable is defined using XML Schema simple type or element, the part attribute MUST NOT be used.

~~**To:** In the first from-spec variant an expression language, identified by the optional expression language attribute, is used to return a value. It allows processes to perform simple computations on properties and variables (for example, increment a sequence number). This value MUST be one of the followings:~~

- ~~a single XML information item other than a character information item (CharII): examples are element information item (EII) and attribute information item (AII)~~
- ~~a sequence of one or more character information items: this is mapped to a Text Node in the data model of XPath 1.0~~

**From:** The fourth ("expression") from-spec variant allows processes to perform simple computations on properties and variables (for example, increment a sequence number).

~~The fifth from-spec variant allows a literal value to be given as the source value to assign to a destination. The type of the literal value MUST be the type of the destination (to-spec). The type of the literal value MAY be optionally indicated inline with the value by using XML Schema's instance type mechanism (xsi:type).~~

~~**To:** The fourth from-spec variant allows a literal value to be given as the source value to assign to a destination. The literal value MUST be treated as a XML fragment that is then serialized into an infoset.~~

**To:** In the fourth from-spec variant, an expression language, identified by the optional expression language attribute, is used to return a value. In the fourth to-spec variant, a

query language, identified by the optional query language attribute, is used to return a value. Both from-spec and to-spec allow processes to perform simple computations on properties and variables (for example, increment a sequence number). This computed value MUST be one of the followings:

- a single XML information item other than a character information item (CharII): examples are element information item (EII) and attribute information item (AII)
- a sequence of one or more character information items: this is mapped to a Text Node in the data model of XPath 1.0

In the case of to-spec, the computed value MUST be a lvalue.

Please note that it is possible to use either the first form of from-spec/to-spec or the fourth form of from-spec/to-spec to perform copy on non-message-variables and parts of message variables, as this specification defines how to manifest non-message variables and parts of message variables as XML Infoset information items. However, only the first form of from-spec/to-spec is able to copy an entire message variable including all of its parts. Other from and to-spec forms are only able to refer to a single part in a WSDL message type variable and so cannot copy all of the parts at once.

**Remove** the paragraph beginning with:

The sixth ("service-ref") from-spec The sixth ("service-ref") from-spec …

# Section 12.5.1

**From:** The expression for a join condition for an activity MUST be constructed using only Boolean operators and the bpws:getLinkStatus function (see Expressions) applied to incoming links at the activity.

**To:** The expression for a join condition for an activity MUST be constructed using only Boolean operators and the control links variable values for the incoming links at the activity.

# Changes in Section "14.1. Expressions"

**From:**
14.1 Expressions

**To:**
9.2.8 BPEL XPath Functions

**From:**

These extensions refer to the **Expressions** feature of BPEL4WS.
The first extension defines a standard fault for erroneous use of the XPath 1.0 function defined for extracting global property values from variables.

**To:**
The XPath 1.0 function getVariableProperty is introduced to allow XPath functions to retrieve property values from variables.

Delete the last 3 paragraphs in the original text, they defined the now unnecessary getVariableData() function.

# Changes in Section "14.3. Assignment"

Replace the entire section with:

The from-spec and to-spec MUST yield a node-set that contains exactly one node. If the from-spec or to-spec selects zero nodes or more than one node during execution, then the standard fault bpws:selectionFailure MUST be thrown by a compliant implementation.

If any of the matching constraints defined in the section Type Compatibility in Assignment is violated during execution, the standard fault bpws:mismatchedAssignmentFailure MUST be thrown by a compliant implementation.

The assign activity is treated as if it were atomic. This means that the assign activity MUST be executed as if, for the duration of its execution, it was the only activity in the process being executed. The mechanisms used to implement the previous requirement are implementation dependent.If there is any fault during the execution of an assignment activity, the destination variables are left unchanged as they were at the start of the activity. This applies regardless of the number of assignment elements within the overall assignment activity.

# References

Insert the following references into the reference section:
[Infoset] XML Information Set (Second Edition), W3C Recommendation

# Fixing Examples

[***Editor's Note: AYIU: a bunch of misc syntax below: from "/" to "." ]

## Section 6.1

**From:**

```
<from variable="PO" part="customerInfo"/>
<to variable="shippingRequest"
  part="customerInfo"/>
```

**To:**

```
<from>$PO/customerInfo</from>
<to>$shippingRequest/customerInfo</to>
```

## Section 8.2 [***Editor's Note: AYIU: Issue 203: Abandon this part of "from-to" changes about propertyAlias]

**From:**

```
        <bpws:propertyAlias propertyName="tns:taxpayerNumber"
               messageType="txmsg:taxpayerInfo" part="identification">
               <query>
               /txtyp:socialsecnumber
               </query>
        </bpws:propertyAlias>
</definitions>
```

**To:**

```
        <bpws:propertyAlias propertyName="tns:taxpayerNumber"
               messageType="txmsg:taxpayerInfo">
               <query>
               identification/txtyp:socialsecnumber
               </query>
        </bpws:propertyAlias>

</definitions>
```

## Section 9.3.2

**From:**

```
<assign>
    <copy>
        <from variable="c1"/>
        <to variable="c2"/>
    </copy>
    <copy>
        <from variable="c1" part = "address"/>
       <to variable="c3"/>
    </copy>
</assign>
```

**To:**

```
<assign>
    <copy>
        <from variable="c1"/>
        <to variable="c2"/>        <from>$c1</from>
        <to>$c2</to>
```

```
    </copy>
    <copy>
        <from>$c1/address</from>
        <to variable="c3"/>        <to>$c3</to>
    </copy>
</assign>
```

## Section 10.2

**From:**

```
<bpws:propertyAlias propertyName="cor:customerID"
        messageType="tns:POMessage" part="PO">
        <bpws:query>
        /PO/CID
        </bpws:query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:orderNumber"
        messageType="tns:POMessage" part="PO">
        <query>
        /PO/Order
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:customerID"
        messageType="tns:POResponse" part="RSP">
        <bpws:query>
        /RSP/CID
        </bpws:query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:orderNumber"
        messageType="tns:POResponse" part="RSP">
        <query>
        /RSP/Order
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:vendorID"
        messageType="tns:POResponse" part="RSP">
        <query>
        /RSP/VID
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:invoiceNumber"
        messageType="tns:POResponse" part="RSP">
        <query>
        /RSP/InvNum
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:vendorID"
        messageType="tns:InvMessage" part="IVC">
        <query>
        /IVC/VID
        </query>
```

```
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:invoiceNumber"
        messageType="tns:InvMessage" part="IVC">
        <query>
        /IVC/InvNum
        </query>
</bpws:propertyAlias>
        ...
</definitions>
```

**To: (Please NOTE: some of the XPaths in the "From-text" are incorrect because the "po:Invoice" element is missing in the XPath for "tns:InvMessage", the "To–text" below includes the correct query language XPaths)**

```
<bpws:propertyAlias propertyName="cor:customerID"
        messageType="tns:POMessage" part="PO">
        <bpws:query>
        CID
        </bpws:query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:orderNumber"
        messageType="tns:POMessage" part="PO">
        <query>
        Order
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:customerID"
        messageType="tns:POResponse" part="RSP">
        <bpws:query>
        CID
        </bpws:query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:orderNumber"
        messageType="tns:POResponse" part="RSP">
        <query>
        Order
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:vendorID"
        messageType="tns:POResponse" part="RSP">
        <query>
        VID
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:invoiceNumber"
        messageType="tns:POResponse" part="RSP">
        <query>
        InvNum
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:vendorID"
        messageType="tns:InvMessage" part="IVC">
        <query>
        VID
        </query>
```

```
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:invoiceNumber"
        messageType="tns:InvMessage" part="IVC">
        <query>
        InvNum
        </query>
</bpws:propertyAlias>


<bpws:propertyAlias propertyName="cor:customerID"
        messageType="tns:POMessage">
        <bpws:query>
        PO/CID
        </bpws:query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:orderNumber"
        messageType="tns:POMessage">
        <query>
        PO/Order
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:customerID"
        messageType="tns:POResponse">
        <bpws:query>
        RSP/CID
        </bpws:query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:orderNumber"
        messageType="tns:POResponse">
        <query>
        RSP/Order
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:vendorID"
        messageType="tns:POResponse">
        <query>
        RSP/VID
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:invoiceNumber"
        messageType="tns:POResponse">
        <query>
        RSP/InvNum
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:vendorID"
        messageType="tns:InvMessage">
        <query>
        IVC/po:Invoice/VID
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="cor:invoiceNumber"
        messageType="tns:InvMessage">
```

```
        <query>
        IVC/po:Invoice/InvNum
        </query>
</bpws:propertyAlias>
        ...
</definitions>
```

## Section 12.3

**From:**
bpws:getVariableData(orderDetails) > 100

**To:**
$orderDetails > 100

## Section 12.5.3

**From:**

```
                bpws:getLinkStatus('buyToSettle') and
                bpws:getLinkStatus('sellToSettle')
```

**To:**

$buyToSettle and $sellToSettle

## Section 13.5.3

**From:**
<for>bpws:getVariableData(orderDetails,processDuration) </for>

**To:**
<for>$orderDetails/.processDuration</for>

## Section 16.1.2

**From:**

```
<bpws:propertyAlias propertyName="tns:shipOrderID"
        messageType="sns:shippingRequestMsg"
        part="shipOrder">
        <query>
        /shipOrder/ShipOrderRequestHeader/shipOrderID
        </query>

</bpws:propertyAlias>

<bpws:propertyAlias propertyName="tns:shipOrderID"
        messageType="sns:shippingNoticeMsg"
        part="shipNotice">
        <query>
        /shipNotice/ShipNoticeHeader/shipOrderID
        </query>

</bpws:propertyAlias>
```

```
<bpws:propertyAlias propertyName="tns:shipComplete"
        messageType="sns:shippingRequestMsg"
        part="shipOrder">
        <query>
        /shipOrder/ShipOrderRequestHeader/shipComplete
        </query>

</bpws:propertyAlias>

<bpws:propertyAlias propertyName="tns:itemsTotal"
        messageType="sns:shippingRequestMsg"
        part="shipOrder">
        <query>
        /shipOrder/ShipOrderRequestHeader/itemsTotal
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="tns:itemsCount"
        messageType="sns:shippingNoticeMsg"
        part="shipNotice">
        <query>
        /shipNotice/ShipNoticeHeader/itemsCount
        </query>

</bpws:propertyAlias>
</wsdl:definitions>
```

**To:**

```
<bpws:propertyAlias propertyName="tns:shipOrderID"
        messageType="sns:shippingRequestMsg"
        part="shipOrder">
        <query>
        ShipOrderRequestHeader/shipOrderID
        </query>

</bpws:propertyAlias>
<bpws:propertyAlias propertyName="tns:shipOrderID"
        messageType="sns:shippingNoticeMsg"
        part="shipNotice">
        <query>
        ShipNoticeHeader/shipOrderID
        </query>

</bpws:propertyAlias>
<bpws:propertyAlias propertyName="tns:shipComplete"
        messageType="sns:shippingRequestMsg"
        part="shipOrder">
        <query>
        ShipOrderRequestHeader/shipComplete
        </query>

</bpws:propertyAlias>
<bpws:propertyAlias propertyName="tns:itemsTotal"
        messageType="sns:shippingRequestMsg"
        part="shipOrder">
        <query>
```

```
        ShipOrderRequestHeader/itemsTotal
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="tns:itemsCount"
        messageType="sns:shippingNoticeMsg"
        part="shipNotice">
        <query>
        ShipNoticeHeader/itemsCount
        </query>

</bpws:propertyAlias>
</wsdl:definitions>


<bpws:propertyAlias propertyName="tns:shipOrderID"
        messageType="sns:shippingRequestMsg">
        <query>
        shipOrder/ShipOrderRequestHeader/shipOrderID
        </query>

</bpws:propertyAlias>

<bpws:propertyAlias propertyName="tns:shipOrderID"
        messageType="sns:shippingNoticeMsg">
        <query>
        shipNotice/ShipNoticeHeader/shipOrderID
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="tns:shipComplete"
        messageType="sns:shippingRequestMsg">
        <query>
        shipOrder/ShipOrderRequestHeader/shipComplete
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="tns:itemsTotal"
        messageType="sns:shippingRequestMsg">
        <query>
        shipOrder/ShipOrderRequestHeader/itemsTotal
        </query>
</bpws:propertyAlias>

<bpws:propertyAlias propertyName="tns:itemsCount"
        messageType="sns:shippingNoticeMsg">
        <query>
        shipNotice/ShipNoticeHeader/itemsCount
        </query>
</bpws:propertyAlias>
</wsdl:definitions>
```

## Section 16.1.3 [***Editor's Note: AYIU: Issue 203: Abandon this part of "from-to" changes about propertyAlias: ###begin]

**From:**

```
<!-- Context type used for locating business process via auction Id -->
        <bpws:property name="auctionId"
                type="xsd:string"/>
```

```
        <bpws:propertyAlias propertyName="tns:auctionId"
               messageType="tns:sellerData"
               part="auctionId"/>
        <bpws:propertyAlias propertyName="tns:auctionId"
               messageType="tns:buyerData"
               part="ID"/>
        <bpws:propertyAlias propertyName="tns:auctionId"
               messageType="tns:auctionData"
               part="auctionId"/>
        <bpws:propertyAlias propertyName="tns:auctionId"
               messageType="tns:auctionAnswerData"
               part="auctionId"/>

<!-- Partner link type for seller/auctionHouse -->
```

**To:**

```
<!-- Context type used for locating business process via auction Id -->
        <bpws:property name="auctionId"
               type="xsd:string"/>
        <bpws:propertyAlias propertyName="tns:auctionId"
               messageType="tns:sellerData">
               <bpws:query>auctionId</bpws:query>
        </bpws:propertyAlias>
        <bpws:propertyAlias propertyName="tns:auctionId"
               messageType="tns:buyerData"
               <bpws:query>ID</bpws:query>
        </bpws:propertyAlias>
        <bpws:propertyAlias propertyName="tns:auctionId"
               messageType="tns:auctionData"
               <bpws:query>auctionId</bpws:query>
        </bpws:propertyAlias>

        <bpws:propertyAlias propertyName="tns:auctionId"
               messageType="tns:auctionAnswerData"
               <bpws:query>auctionId</bpws:query>
        </bpws:propertyAlias>
<!-- Partner link type for seller/auctionHouse -->
```

[***Editor's Note: AYIU: Issue 203: Abandon this part of "from-to" changes about propertyAlias: ###end]

**From:**

```
<from><expression>"0"</expression></from>
 <to variable="itemsShipped"/>
```

**To:**

```
 <from>0</from>
 <to>$itemsShipped</to>
```

**From:**

bpws:getVariableData('itemsShipped') < bpws:getVariableProperty ('shipRequest','props:itemsTotal')

**To:**

$itemsShipper < bpws:getVariableProperty('shipRequest','props:itemsTotal')

**From:**

```
                        <from>
                          <expression>bpws:getVariableData
('itemsShipped')+
                                        bpws:getVariableProperty
('shipNotice','props:itemsCount')
                          </expression>
                        </from>
                        <to variable="itemsShipped"/>
```

**To:**

```
                        <from>$itemsShipped +bpws:getVariableProperty
('shipNotice',
                        'props:itemsCount')</from>
                        <to>$itemsShipped</to>
```

## Section 16.2.2

**From:**

```
bpws:getVariableData('request','amount')&lt; 10000
```

**To:**

```
$request/.amount &lt; 10000
```

**From:**

```
bpws:getVariableData('request','amount')>=10000
```

**To:**

```
$request./amount >= 10000
```

**From:**

```
bpws:getVariableData('risk','level')='low'
```

**To:**

```
$risk/.level = 'low'
```

**From:**

```
bpws:getVariableData('risk','level')!='low'
```

**To:**

```
$risk/.level != 'low'
```

## Section 16.3.2

**From:**

```
            <to variable="auctionData"
                  part="auctionHouseServiceRef"/>
```

**To:**

<to>$auctionData/.auctionHouseServiceRef</to>

**From:**

<from variable="sellerData"
part="endpointReference"/>

**To:**

<from>$sellerData/.endpointReference</from>

**From:**

<from variable="buyerData"
part="endpointReference"/>

**To:**

<from>$buyerData/.endpointReference</from>

**From:**

```
<from>
  <service-ref>
    <addr:EndpointReference>
      <addr:Address>xs:anyURI</addr:Address>
      <addr:ServiceName>as:RegistrationService</addr:ServiceName>
    </addr:EndpointReference>
  </service-ref>
</from>
```

**To:**

```
<from literal=”yes” >
  <literal>
  <service-ref>
    <addr:EndpointReference>
      <addr:Address>xs:anyURI</addr:Address>
      <addr:ServiceName>as:RegistrationService</addr:ServiceName>
    </addr:EndpointReference>
  </service-ref>
  </literal>
</from>
```

# Schema Update

## In wsbpel_main.xsd:

**From:**

```
    <complexType name="tFrom">
        <complexContent>
            <extension base="bpws:tExtensibleElements">
                <sequence>
                    <choice minOccurs="0">
                        <element ref="bpws:query"/>
                        <element ref="bpws:expression"/>
```

```
                        <element ref="bpws:service-ref"/>
                    </choice>
                </sequence>
                <attribute name="variable" type="NCName"/>
                <attribute name="part" type="NCName"/>
                <attribute name="property" type="QName"/>
                <attribute name="partnerLink" type="NCName"/>
                <attribute name="endpointReference" type="bpws:tRoles"/>
                <attribute name="opaque" type="bpws:tBoolean"/>
            </extension>
        </complexContent>
    </complexType>
```

**To:**

```
        <complexType name="tFrom" mixed="true">
            <sequence>
                <element ref="bpws:documentation" minOccurs="0"
maxOccurs="unbounded"/>
                <any namespace="##other" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
                <choice minOccurs="0">
                    <element ref="bpws:service-ref"/>
                </choice>
            </sequence>
            <attribute name="literal" type="bpws:tBoolean"/>
            <attribute name="expressionLanguage" type="anyURI"/>
            <attribute name="variable" type="NCName"/>
            <attribute name="property" type="QName"/>
            <attribute name="partnerLink" type="NCName"/>
            <attribute name="endpointReference" type="bpws:tRoles"/>
            <attribute name="opaque" type="bpws:tBoolean"/>
            <anyAttribute namespace="##other"
processContents="lax"/>
        </complexType>

        <complexType name="tFrom" mixed="true">
            <sequence>
                <element ref="bpws:documentation" minOccurs="0"
maxOccurs="unbounded"/>
                <any namespace="##other" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
                <choice minOccurs="0">
                    <element name="literal">
                        <complexType mixed="true">
                            <sequence>
                                <any
namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
                            </sequence>
                        </complexType>
                    </element>
                </choice>
            </sequence>
            <attribute name="expressionLanguage" type="anyURI"/>
            <attribute name="variable" type="NCName"/>
             <attribute name="part" type="NCName"/>
            <attribute name="property" type="QName"/>
            <attribute name="partnerLink" type="NCName"/>
            <attribute name="endpointReference" type="bpws:tRoles"/>
            <attribute name="opaque" type="bpws:tBoolean"/>
            <anyAttribute namespace="##other"
processContents="lax"/>
```

```
        </complexType>
```

Delete the complexType “tQuery” and the element “query”
Delete the element “expression”.

**From:**

```
    <element name="to">
        <complexType>
            <complexContent>
                <restriction base="bpws:tFrom">
                    <sequence>
                        <element ref="bpws:documentation" minOccurs="0"
maxOccurs="unbounded" />
                        <any namespace="##other" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
                        <choice minOccurs="0">
                            <element ref="bpws:query"/>
                        </choice>
                    </sequence>
                    <attribute name="opaque" type="bpws:tBoolean"
use="prohibited"/>
                    <attribute name="endpointReference"
type="bpws:tRoles" use="prohibited"/>
                </restriction>
            </complexContent>
        </complexType>
    </element>
```

**To:**

```
        <element name="to">
            <complexType mixed="true">
                <sequence>
                    <element ref="bpws:documentation"
minOccurs="0" maxOccurs="unbounded"/>
                    <any namespace="##other"
processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
                </sequence>
                <attribute name="queryLanguage" type="anyURI"/>
                <attribute name="variable" type="NCName"/>
                <attribute name="property" type="QName"/>
                <attribute name="partnerLink" type="NCName"/>
                <anyAttribute namespace="##other"
processContents="lax"/>
            </complexType>
        </element>

        <element name="to" type="bpws:tTo" />
        <complexType name="tTo" mixed="true">
                <sequence>
                    <element ref="bpws:documentation"
minOccurs="0" maxOccurs="unbounded"/>
                    <any namespace="##other"
processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
                </sequence>
                <attribute name="queryLanguage" type="anyURI"/>
                <attribute name="variable" type="NCName"/>
                <attribute name="part" type="NCName"/>
                <attribute name="property" type="QName"/>
                <attribute name="partnerLink" type="NCName"/>
```

```
                        <anyAttribute namespace="##other"
processContents="lax"/>
        </complexType>
```

**In msgprop.xsd:** [***Editor's Note: AYIU: Issue 203: Abandon this part of "from-to" changes about propertyAlias: ###end]

**From:**

```
<element name="propertyAlias">
  <complexType>
    <sequence>
      <element name="query" minOccurs="0">
        <complexType mixed="true">
          <sequence>
            <any processContents="lax" minOccurs="0"/>
          </sequence>
          <attribute name="queryLanguage" type="anyURI"/>
        </complexType>
      </element>
    </sequence>
    <attribute name="propertyName" type="QName" use="required"/>
    <attribute name="messageType" type="QName" use="required"/>
    <attribute name="part" type="NCName"/>
  </complexType>
</element>
```

**To:**

We can drop the "part" attribute.

**END**