A WF Xaml Vocabulary Specification

This document contains an inofficial *Microsoft Windows Workflow Xaml Vocabulary* [MS-WFXV] specification used in the tool WF2oWFN[1]. It is not intended to be complete nor self–contained, but does currently only cover activities and their control–flow relevant properties implemented in the tool. The specification captures WF's Xaml vocabulary as of .NET 4.*x*. The general XAML Object Mapping Specification [Mic12a] is available online. Since the referenced general specification was not available at the time of the initial creation of this document, we use a different syntax to define the grammar of WF's Xaml derivation than other official vocabularies do[2].

A.1 Notational Conventions

For grammar definition, we use an informal syntax similar to the one used in the BPEL specification [OAS07]. The syntax is presented like a normal XML instance, though the values describe data types rather than concrete data instances. Capitalized expressions in navy color are references to basic data types (e.g. `QNAME`). Black emphases symbolize references to an external definition (e.g. `standard-attributes`). XamlTypes (elements) are formatted in cyan (e.g. `WriteLine`), XamlMembers (attributes) in purple (e.g. `DisplayName`). Moreover, quantifiers can be attached to elements and attributes. If no quantifier is specified, the element has to exist exactly once. '`?`' signifies a unique optional (0 or 1), '`*`' a repeated optional occurence (0 or more). Elements and attributes with the quantifier '`+`' have to exist at least once (1 or more). The characters '`[`' and '`]`' are used to indicate that the contained items are to be treated as a group. Elements and attributes separated by '`|`' and grouped by "'`(`"' and "'`)`"' are syntactic alternatives. Elements and attributes on the same scope are not ordered, i.e. they can occur in an arbitrary arrangement. The described snippets are fragments and require additional information in order to be executable. Every activity has to be included within an `Activity` root node (for *.xaml files) or a `WorkflowService` root node (for *.xamlx files), which also includes all referenced namespaces. Within the definition several namespaces are used to qualify elements and attributes. The choice of prefixes in this specification is arbitrary, non–normative and semantically not significant. The specification currently focuses on a specific set of activities implemented in the compiler WF2oWFN. Potential minor impacts by other not specified activities cannot be ruled out. The specification was determined by extensive unit tests and with reference to the .NET WF API [Mic12b, Mic12c]. Missing an official specification it may be possible that, despite extensive testing, additional syntactic variants exist.

A.2 WF Xaml Schema

Below, the used *Schema Information Items* are listed. These define the root of the schema definition, including all other items that belong to this schema [Mic12a, § 3]. This specification defines or makes use of three different Xaml Schema Information Items:

[1] Available at `https://github.com/uniba-dsg/wf2owfn`

[2] Publicly available vocabularies like Silverlight or Windows Presentation Foundation (WPF).

The 'Activities' Schema

Property	Value
[spec ns prefix]	`xmlns`
[target namespace]	http://schemas.microsoft.com/netfx/2009/xaml/activities
[types]	See section A.3

The 'Servicemodel' Schema

Property	Value
[spec ns prefix]	`xmlns:p`
[target namespace]	http://schemas.microsoft.com/netfx/2009/xaml/servicemodel
[types]	See section A.3.1

The 'x' Schema[3]

Property	Value
[spec ns prefix]	`xmlns:x`
[target namespace]	http://schemas.microsoft.com/winfx/2006/xaml
[types]	See [Mic12a, § 5]

Subsequently, further general definitions are introduced:

`QNAME` = Qualified name, consisting of namespace and local name.
`NCNAME` = Simple literal.
`BOOL` = Simple boolean value.

`activity` = An arbitrary activity from the 'Activities' or 'Servicemodel' schema, i.e. the corresponding WF API class implements `System.Activities.Activity` (see [Mic12b, Mic12c]). Also indicated with an attached `Activity` flag in the specification.

`namespace` = An arbitrary namespace declaration.

A.3 WF XamlType Information Items

Workflow Declaration

WF supports two different possibilities to define Xaml–based workflows. One the one hand standard `Activity` workflows on the other hand service–based `WorkflowServices`.

```
<Activity x:Class="NCNAME" namespace+>
   activity
</Activity>
```

```
<p:WorkflowService Name="NCNAME" ConfigurationName="QNAME" namespace+>
   activity
</p:WorkflowService>
```

[3] Intrinsic Schema Information Item that contains directives and types available across all Xaml vocabularies.

Both attributes *x:Name* and *x:Key* can exist in every `activity` and additionally in the control–flow constructs of a flowchart (i.e. `FlowDecision`, `FlowSwitch<T>`). *x:Name* contains a unique identifier in the form `__ReferenceID1`, which can be used to reference an activity in the control–flow. The value of the attribute *x:Key* identifies the covered case of a `Switch<T>` or `FlowSwitch<T>` activity.

`standard-attributes`

```
x:Name="NCNAME"? , x:Key="QNAME"?
```

`reference` = An attribute–based reference of an activity. This is realized with a type–value combination (e.g. `{x:Reference NCNAME}`).

`vb-expr` = A Visual Basic expression, which has to evaluate to an expected type.

Attribute Syntax (Properties) and Property Element Syntax[4]

XamlTypes have the ability to express the same configuration both as an attribute and as a *Property Element*. In any case only one of both variants can be used. In Xaml, objects can be assigned as properties. This possibility is described with the Property Element syntax. The property is not declared inside the element tag as an attribute but with the aid of an element in the form *$ActivityName.$PropertyName*, where both variables have to be replaced by a specific instance name.

```
<$ActivityName>
   <$ActivityName.$PropertyName>
      Serialized object manifestation
   </$ActivityName.$PropertyName>
</$ActivityName>
```

For such objects this technique is mandatory. Primitive types, like literal properties, can be described both as an attribute and as Property Element, even if it is not recommended to use the latter in this case.

Representation of a primitive type as an attribute:

```
<$ActivityName Text="NCNAME" />
```

Equivalent representation of a primitive type as a Property Element:

```
<$ActivityName>
   <$ActivityName.Text> NCNAME </$ActivityName.Text>
</$ActivityName>
```

Hereafter, only actually observed syntax manifestation of WF are displayed in the specification.

Variables

Many activities support the declaration of variables in their scope. *$ActivityName* is again to be replaced by a specific activity name.

[4] See `http://msdn.microsoft.com/en-US/library/vstudio/ms752059(v=vs.100).aspx`

variables

```
<$ActivityName.Variables>
   <Variable
      x:TypeArguments="QNAME"
      Name="NCNAME"
      Default="vb-expr"? />+
</$ActivityName.Variables>
```

Input and output arguments

Dataflow in and out of activities can be defined with arguments.

inargument

```
<InArgument x:TypeArguments="QNAME"> vb-expr </InArgument>
```

outargument

```
<OutArgument x:TypeArguments="QNAME"> vb-expr </OutArgument>
```

A.3.1 Primitive Activities

WriteLine Activity

```
<WriteLine
   Text="vb-expr"?
   TextWriter="vb-expr"?
   DisplayName="NCNAME"?
   standard-attributes />
```

Expected types:
Text = System.String, TextWriter = System.IO.TextWriter

Delay Activity

```
<Delay
   Duration="vb-expr"
   DisplayName="NCNAME"?
   standard-attributes />
```

Expected types: Duration = System.Timespan

Assign Activity

```
<Assign DisplayName="NCNAME"? standard-attributes>
   <Assign.To> out-argument </Assign.To>
   <Assign.Value> in-argument </Assign.Value>
</Assign>
```

Receive Activity

Ellipsis signal further complex inner structures (e.g. parameters, correlations). As data aspects were abstracted in the approach, we refrain from an exact definition.

```
<p:Receive
   OperationName="NCNAME"
   ServiceContractName="QNAME"?
   Action="NCNAME"?
   CanCreateInstance="BOOL"?
   SerializerOption="DataContractSerializer | XmlSerializer"?
   CorrelatesWith="vb-expr"?
   ProtectionLevel="None | Sign | EncryptAndSign"?
   DisplayName="NCNAME"?
   standard-attributes>
   (...)
</p:Receive>
```

Expected types:
CorrelatesWith = System.ServiceModel.Activities.CorrelationHandle

ReceiveReply Activity

See A.3.1.

```
<p:ReceiveReply
   Request="reference"
   Action="NCNAME"?
   DisplayName="NCNAME"?
   standard-attributes>
   (...)
</p:ReceiveReply>
```

Send Activity

See A.3.1.

```
<p:Send
   OperationName="NCNAME"
   ServiceContractName="QNAME"
   Action="NCNAME"?
   CorrelatesWith="vb-expr"?
   EndpointAddress="vb-expr"?
   EndpointConfigurationName="NCNAME"?
   ProtectionLevel="Sign | None | EncryptAndSign"?
   SerializerOption="DataContractSerializer | XmlSerializer"?
   TokenImpersonationLevel="None | Anonymous | Identification |
      Impersonation | Delegation"?
   DisplayName="NCNAME"?
   standard-attributes>
   (...)
</p:Send>
```

Expected types:
CorrelatesWith = System.ServiceModel.Activities.CorrelationHandle,
EndpointAddress = System.Uri

SendReply Activity

See A.3.1.

```
<p:SendReply
   Request="reference"
   Action="NCNAME"?
   PersistBeforeSend="BOOL"?
   DisplayName="NCNAME"?
   standard-attributes>
   (...)
</p:SendReply>
```

A.3.2 Structured Activities

Sequence Activity

```
<Sequence DisplayName="NCNAME"? standard-attributes>
   variables ?
   activity *
</Sequence>
```

Parallel Activity

Only one of both possible representations of the attribute *CompletionCondition* exists (See A.3).

```
<Parallel CompletionCondition="vb-expr"? DisplayName="NCNAME"?
   standard-attributes>
   variables ?
   <Parallel.CompletionCondition>?
      vb-expr
   </Parallel.CompletionCondition>
   activity *
</Parallel>
```

Expected types:
CompletionCondition = System.Boolean

Pick Activity

```
<Pick DisplayName="NCNAME"? standard-attributes>
   <PickBranch DisplayName="NCNAME"?>*
      variables ?
      <PickBranch.Trigger>
         activity
      </PickBranch.Trigger>
      activity ?
   </PickBranch>
</Pick>
```

While Activity

Only one of both possible representations of the attribute *Condition* exists (See A.3).

```
<While Condition="vb-expr" DisplayName="NCNAME"? standard-attributes>
   variables ?
   <While.Condition>
      vb-expr
   </While.Condition>
   activity ?
</While>
```

Expected types:
Condition = System.Boolean

DoWhile Activity

Only one of both possible representations of the attribute *Condition* exists (See A.3).

```
<DoWhile Condition="vb-expr" DisplayName="NCNAME"? standard-attributes>
   variables ?
   <DoWhile.Condition>
      vb-expr
   </DoWhile.Condition>
   activity ?
</DoWhile>
```

Expected types:
Condition = System.Boolean

If Activity

```
<If Condition="vb-expr" DisplayName="NCNAME"? standard-attributes>
   <If.Then>?
      activity
   </If.Then>
   <If.Else>?
      activity
   </If.Else>
</If>
```

Expected types:
Condition = System.Boolean

Switch<T> Activity

All inner activities additionally contain the attribute *x:Key* to denote the handled case. The *QNAME* of the *x:Key* attribute can also be a simple literal for primitive data types. The element `<x:Null>` exists if no activity was assigned for the defined case.

```
<Switch
   x:TypeArguments="QNAME"
   Expression="vb-expr"
   DisplayName="NCNAME"?
   standard-attributes>
   <Switch.Default>?
      activity
   </Switch.Default>
   <x:Null x:Key="QNAME" />*
   activity *
</Switch>
```

Expected types:
Expression = T

A.3.3 Flowchart Activity

Only one of both possible representations of the attribute *StartNode* exists (See A.3). If the flowchart contains no activities, i.e. it is empty, the node `<Flowchart.StartNode>` contains the element `<x:Null />`. If any of the flowchart's inner structures is not linked to another node, the flowchart terminates on that regard.

```
<Flowchart StartNode="reference" DisplayName="NCNAME"?
   standard-attributes>
   variables ?
   <Flowchart.StartNode>
      flownode | reference
   </Flowchart.StartNode>
   flownode *
</Flowchart>
```

flownode

```
flowstep | flowdecision | flowswitch | ref-element
```

flowstep

```
<FlowStep standard-attributes>
   activity
   <FlowStep.Next>?
      flownode
   </FlowStep.Next>
</FlowStep>
```

ref-element

```
<x:Reference>
   NCNAME
   <x:Key>?
      QNAME
   </x:Key>
</x:Reference>
```

FlowDecision

Only one of both possible representations of the attributes *True, False* exists (See A.3).

```
<FlowDecision Condition="vb-expr" True="reference"? False="reference"?
   standard-attributes>
   <FlowDecision.True>?
      flownode
   </FlowDecision.True>
   <FlowDecision.False>?
      flownode
   </FlowDecision.False>
</FlowDecision>
```

Expected types:
Condition = System.Boolean

FlowSwitch<T>

All referenced activities of `FlowSwitch<T>` or an inner `ref-element` additionally contain the attribute *x:Key* to denote the handled case. Only one of both possible representations of the attribute *Default* exists (See A.3).

```
<FlowSwitch Default="reference"? Expression="vb-expr"
   standard-attributes>
   <FlowSwitch.Default>?
      flownode
   </FlowSwitch.Default>
   flownode *
</FlowSwitch>
```

Expected types:
Expression = T

A.3.4 StateMachine Activity

Only one of both possible representations of the attribute *InitialState* exists (See A.3).

```
<StateMachine
   InitialState="reference"? DisplayName="NCNAME"?>
   variables ?
   (
   state
   <x:Reference>*
      NCNAME
   </x:Reference>
   |
   <StateMachine.InitialState>
      state
   </StateMachine.InitialState>
   <x:Reference>*
      NCNAME
   </x:Reference>
   )
</StateMachine>
```

If *isFinal* is true, then the element `State` must only contain the element `State.Entry`.

`state`

```
<State x:Name="NCNAME" isFinal="BOOL"? DisplayName="NCNAME"?>
   variables ?
   <State.Entry>?
      activity
   </State.Entry>
   <State.Exit>?
      activity
   </State.Exit>
   <State.Transitions>?
      transition +
   </State.Transitions>
</State>
```

Each `State` can only contain one `Transition` without *Condition* and *Trigger*. In case of a *Shared Trigger*, which activity triggers several transitions, the respective trigger activity is referenced with the aid of the attribute *x:Name*. Again, only one of both possible representations of the attributes *Condition, To, Trigger* exists (See A.3).

`transition`

```
<Transition To="reference" Trigger="reference"? Condition="vb-expr"?>
   <Transition.Trigger>?
      activity
   </Transition.Trigger>
   <Transition.To>?
      state | <x:Reference>NCNAME</x:Reference>
   </Transition.To>
   <Transition.Action>?
      activity
   </Transition.Action>
   <Transition.Condition>?
      vb-expr
   </Transition.Condition>
</Transition>
```

Expected types:
`Condition` = `System.Boolean`

A.4 Version History

Version 1.1 (19 October 2012)

- Added several terms and references to Xaml Object Mapping Specification [Mic12a]

Version 1.0 (29 June 2012)

- Initial version covering activities of WF2oWFN v0.1

References

[Mic12a] Microsoft: *[MS-XAML]: Xaml Object Mapping Specification 2009*, April 2012. Available online at `http://download.microsoft.com/download/0/A/6/0A6F7755-9AF5-448B-907D-13985ACCF53E/[MS-XAML-2012].pdf`; last accessed on June 29th 2012.

[Mic12b] Microsoft: *System.Activities.Statements Namespace*, 2012. Available online at `http://msdn.microsoft.com/en-us/library/System.Activities.Statements`; last accessed on June 29th 2012.

[Mic12c] Microsoft: *System.ServiceModel.Activities Namespace*, 2012. Available online at `http://msdn.microsoft.com/en-us/library/system.servicemodel.activities`; last accessed on June 29th 2012.

[OAS07] OASIS: *Web Services Business Process Execution Language Version 2.0*, April 2007. Available online at `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`; last accessed on June 29th 2012.