NATO Communications and Information Agency
Agence OTAN d'information et de communication

AGENCY INSTRUCTION

INSTR TECH 06.02.09

SERVICE INTERFACE PROFILE FOR A PUBLISH/SUBSCRIBE NOTIFICATION BROKER WITH SUBSCRIPTION MANAGER

Effective date:

Revision No: Original

Issued by: Chief, Core Enterprise Services

Approved by: Director Service Strategy

Table of Changes and Amendments

Amendment No	Date issued	Remarks
Ver. 1.0	13th June 2014	*[A. Gruler]* Updated requirements for filter semantics (in comparision to the original TR/2012/SPW008000/24)

Author Details

Organization	Name	Contact Email/Phone
NCI Agency	V. de Sortis	vincenzo.desortis@ncia.nato.int
NCI Agency	M. Lehmann	marek.lehmann@ncia.nato.int
NCI Agency	R. Fiske	rui.fiske@ncia.nato.int
NCI Agency	L. Schenkels	leon.schenkels@ncia.nato.int
NCI Agency	G. Gujral	davinder.gujral@ncia.nato.int

Table of Contents

List of Annexes

AGENCY INSTRUCTION 06.02.09

SERVICE INTERFACE PROFILE FOR A PUBLISH/SUBSCRIBE NOTIFICATION BROKER WITH SUBSCRIPTION MANAGER

0 PRELIMINARY INFORMATION

0.1 References

A. NCIA/GM/2012/235; Directive 1 Revision 1; dated 3 May 2013
B. NCIARECCEN-4-22852 DIRECTIVE 01.01 Agency Policy on Management and Control of Directives, Notices, Processes, Procedures and Instructions, dated 20 May 2014
C. NCIARECCEN-4-23297, Directive 06.00.01, Management and Control of Directives, Processes, Procedures and Instructions on Service Management, dated 03 June 2014

0.2 Purpose

This Technical Instruction (TI) provides detailed information, guidance, instructions, standards and criteria to be used when planning, programming, and designing Agency products and services. In this specific case the TI defines a Service Interface Profile (SIP) for one of NATO's Core Enterprise Services.

TIs are living documents and will be periodically reviewed, updated, and made available to Agency staff as part of the Service Strategy responsibility as Design Authority. Technical content of these instructions is the shared responsibility of SStrat/Service Engineering and Architecture Branch and the Service Line of the discipline involved.

TIs are primarily disseminated electronically[1], and will be announced through Agency Routine Orders. Hard copies or local electronic copies should be checked against the current electronic version prior to use to assure that the latest instructions are used.

0.3 Applicability

This TI applies to all elements of the Agency, in particular to all NCI Agency staff involved in development of IT services or software products. It is the responsibility of all NCI Agency Programme, Service, Product and Project Managers to ensure the implementation of this technical instruction and to incorporate its content into relevant contractual documentation for external suppliers.

1 INTRODUCTION

This document is part of a Service Interface Profile (SIP) for Publish/Subscribe Core Enterprise Services (CES) and should be read together with the main document [NCIA AD 06.05.04.02.E]. It gives guidance on implementation of a WS-Notification compliant *notification broker*. It is REQUIRED that each *notification broker* implementation also includes the *subscription manager* functionality.

2 SIP DEFINITION

2.1 Subject

This SIP focuses on the interface profile of a WS-Notification *notification broker/subscription manager* (henceforth indicated with the term *"notification broker"*). It is part of the Publish/Subscribe services as defined in the CES Framework [NAC AC/322(SC/1)N(2009)0015 (INV), 2009].

[1] https://servicestrategy.nr.ncia/SitePages/Agency%20Directives%20(Technical).aspx

This SIP identifies and describes the mandatory service interfaces needed for basic interoperability.

2.2 Service Interface

The following rules are valid for the *notification broker/subscription manager* interface:

- A *notification broker* MUST support the `NotificationProducer` interface as prescribed in Section 4 of [OASIS WS-BaseNotification, 2006] and Section 5 of [OASIS WS-BrokeredNotifcation, 2006]. The REQUIRED operations to implement are:
  - `Subscribe` – to create a new subscription
  - `GetCurrentMessage` – to return the last *notification* published to a given topic.
- A *notification broker* SHOULD support the *pull-style notification* interface as prescribed in Section 5 of [OASIS WS-BaseNotification, 2006] and Section 5 of [OASIS WS-BrokeredNotification, 2006]. This is more restrictive than in the [OASIS WS-BrokeredNotification, 2006] specification, which leaves this interface optional. The RECOMMENDED operations to implement are:
  - `CreatePullPoint` – to create a new *pull point*
  - `DestroyPullPoint` – to destroy a *pull point*
  - `GetMessages` – to return the *notification* published to a *pull point*.
- A *notification broker* MUST support the `RegisterPublisher` interface as prescribed in Section 6 of [OASIS WS-BrokeredNotification, 2006]. The REQUIRED operation to implement is:
  - `RegisterPublisher` – to create a new publisher registration.
- A *notification broker* SHOULD support the `PublisherRegistrationManager` interface as prescribed in Section 7 of [OASIS WS BrokeredNotification, 2006]. This is more restrictive than in the [OASIS WS-BrokeredNotification, 2006] specification, which leaves this interface optional. The RECOMMENDED operation to implement is:
  - `DestroyRegistration` – to destroy a publisher registration.
- A *notification broker* MUST support the `NotificationConsumer` interface as prescribed in Section 3 of [OASIS WS-BaseNotification, 2006], Section 5 of [OASIS WS-BrokeredNotification, 2006] and *notification consumer* SIP. The REQUIRED operation to implement is:
  - `Notify` – a *notification broker* MUST implement the notify message exchange from the `NotificationConsumer` interface in [OASIS WS-BrokeredNotification, 2006].
- A *notification broker* MUST support the `SubscriptionManager` interface as prescribed in Sections 6 and 6.1 of [OASIS WS-BaseNotification, 2006]. This is an addition to the mandatory interfaces identified in Section 5 of [OASIS WS-BrokeredNotification, 2006]. The REQUIRED operations to implement are:
  - `Renew` – to renew a subscription
  - `Unsubscribe` – to terminate a subscription.

  OPTIONAL operations are:
  - `Pause subscription` – to pause a subscription
  - `Resume subscription` – to resume a paused subscription.

2.3 Behaviours not specified in the WS-Notification Standard

Although the WS-Notification specification is quite complete, there are some possible scenarios that are not described. The goal of this section is to fill this gap.

2.3.1 Subscription and Data Persistence

The WS-Notification specification is not meant to describe or mandate the implementation details. However, in some situations additional implementation rules and behaviours are relevant to guarantee reliable operations.

In order to prevent losses in the case of system/application restarts, and in addition to what is stated in the specifications, it is RECCOMENDED that the *notification broker* and *notification producer*:

- Persistently store information on subscriptions.
- Are able to re-initiate valid subscriptions after a restart.
- Persistently store the notifications received from notification publishers/producers but not yet sent to notification consumers.
- Are able to correctly complete the processing of the received but not sent *notifications* after restart.

2.3.2 Notification Consumer Unreachable

The notification broker (acting as notification producer) produces notifications for those notification consumers for which subscriptions have been registered. If the notification consumer end-point appears to be unreachable, the notification producer:

- MUST retry to send the incoming and not delivered *notifications*. The number and the frequency of retries are not mandated, but MUST be configurable.
- If the retry fails, SHOULD accumulate the incoming *notifications* that have not yet been delivered for a delayed retry.
- SHOULD advertise the number and the frequency of retries through policy assertions or other means.

In case the *notifications* are accumulated, the *notification broker*:

- SHOULD store the accumulated *notifications* in a persistent set to prevent losses in case of system/application restart.
- SHOULD advertise through policy assertions or other means the behaviour in case the queue is filled up.
- SHOULD advertise through policy assertions or other means the notification retention time.

In case the *notifications* are accumulated and if the *notification producer* is no longer able to store them, it:

- MAY ignore the most recent *notifications*, or
- MAY dispose of any previously accumulated *notifications* and continue to accumulate the incoming.

The *notification producer* SHOULD advertise the exposed behaviour through policy assertions or other means.

2.3.3 Pull Points

If the pull-style notification interface is supported, the notification broker:

- MUST guarantee the *pull-point* persistence across application or system restarts.
- MUST guarantee the persistence of the *notification* in the *pull-point* queue across application or system restart.
- SHOULD advertise through policy assertions or other means the behaviour in case the *pull-point* queue is no longer able to hold the *notification*.

In case the *pull-point* queue is no longer capable to hold the *notification*, it:

- MAY ignore the most recent *notifications,* or
- MAY dispose of any previously accumulated *notifications* and continue to accumulate the incoming.

There are situations in which a *pull point* can be considered unused, for example if the messages accumulated at the *pull point* are not retrieved by the *notification consumer* for a long time or if the *pull point* is not subscribed to any *topic*. In such situations the *notification broker* can decide to apply specific strategies to terminate the *pull point* or to pause the subscriptions to which the *pull point* is subscribed.

The *notification broker* SHOULD advertise how it discovers and handles the unused *pull points* through policy assertions or other means.

The *notification broker* SHOULD advertise through policy assertions or other means the order in which messages will be delivered by a `GetMessages` operation.

The *notification broker* SHOULD advertise through policy assertions or other means what happens with messages not delivered because they are above the `MaximumNumber` parameter of a `GetMessages` operation.

2.3.4 Combination of Multiple Filter Elements

The `wsnt:Filter` element can contain multiple sub-elements, such as `wsnt:TopicExpression`, `wsnt:MessageContent`, etc, where each one restricts the set of *notifications* that should be retrieved in a specific way. The WS Notifcation Specification [OASIS WS-BaseNotification, 2006] does not precisely specify the semantics for the case when more than one sub-elements exist as child elements of a `wsnt:Filter` element.

In line with WS Notifcation Specification [OASIS WS-BaseNotification, 2006], if exactly one child element is present the *Notification Broker* MUST only return those *Notifications* for which the corresponding expression evaluates to TRUE, following the evaluation semantics specified for the corresponding child element.

If multiple child elements are present the *Notification Broker* has the freedom to decide on the semantics in which it combines the expressions of the child elements, using for example an AND, or OR semantics. However, the *Notification Broker* SHALL exactly specify the rules which it uses when combining multiple filter/child elements, and how this controls the resulting set of *Notifications* that are returned in response to such a request that contains multiple filter expressions. The *Notification Broker* SHALL publish the specification of the combination rules to its service consumers. RECOMMENDED ways of publishing are via policy assertions, meta-data, or by other means.

2.4 Operations

2.4.1 Operation "Subscribe"

This is a request-response operation initiated from the *subscriber* that sends a subscribe message to a *notification broker*. Creating a subscription the *subscriber* registers the interest of a *notification consumer* to receive a set of *notifications*.

The *notification broker* acting as *notification producer* MUST implement the `subscribe` operation of the `NotificationProducer` interface as defined in Section 4.2 of [OASIS WS-BaseNotification, 2006] and Section 5.3 of [OASIS WS-BrokeredNotification, 2006].

The *notification broker* SHOULD implement at least the following `TopicExpression` dialects for the `/wsnt:Subscribe/wsnt:Filter/wsnt:TopicExpression/@Dialect`:

- Simple `TopicExpression` dialect:

  `http://docs/oasis-open.org/wsn/t-1/TopicExpression/Simple` as defined in Section 8.1 of [OASIS WS-Topics, 2006].

- Concrete `TopicExpression` dialect:

  `http://docs/oasis-open.org/wsn/t-1/TopicExpression/Concrete` as defined in Section 8.2 of [OASIS WS-Topics, 2006].

The *notification broker* MAY support any other `TopicExpression` dialect, including other dialects defined in the [OASIS WS-Topics, 2006] specification.

The *notification broker* SHOULD support [W3C Xpath, 1999] 1.0 as a `MessageContent` filter expression dialect:

- `/wsnt:Subscribe/wsnt:Filter/wsnt:MessageContent/@Dialect=http://www.w3.org/TR/1999/REC-xpath-19991116`.

The *notification broker* MAY support any other filter expression dialect, including custom-expression dialects.

Every subscription message sent to the *notification broker* SHOULD contain `/wsnt:Subscribe/wnst:InitialTerminationTime`. The *notification broker* SHOULD specify via policy assertions, metadata or by some other means its behaviour if the `InitialTerminationTime` element is omitted. It is RECOMMENDED for all *subscriptions* to be defined for a finite duration and be periodically renewed if necessary. It is NOT RECOMMENDED to define subscriptions with infinite durations.

The *notification broker* MUST be able to produce *notifications* using a `wsnt:notify` wrapper. It may also be able to produce the notification messages in a raw form. For this reason the *notification broker*:

- MUST understand subscription messages without any `/wsnt:Subscribe/wsnt:SubscriptionPolicy/wsnt:UseRaw` element.
- MAY understand subscription messages with a single `/wsnt:Subscribe/wsnt:SubscriptionPolicy/wsnt:UseRaw` element.

2.4.1.1 Extensions to the [OASIS WS-BaseNotification, 2006] specification

A subscription message MAY contain the following additional element:

- `/wsnt:Subscribe/wsn-nato:SubscriptionEndTopic`

A `TopicExpression` describing exactly one *topic* which is the *topic* associated with the `SubscriptionEnd` *notification*.

- `/wsnt:Subscribe/wsn-nato:SubscriptionEndTopic/@Dialect`

  The dialect used in the `TopicExpression`. It is RECOMMENDED to use one of the `TopicExpression` dialects described in [OASIS WS-Topics, 2006].

The *notification broker* SHOULD understand the element:

- `/wsnt:Subscribe/wsn-nato:SubscriptionEndTopic`

  When the `/wsnt:Subscribe/wsn-nato:SubscriptionEndTopic` element is present, the *notification broker* SHOULD be able to create, in the case of an unexpected subscription termination, a new `SubscriptionEnd` message and notify it with the *topic* indicated in the `/wsnt:Subscribe/wsn-nato:SubscriptionEndTopic` element to any end-point subscribed for that *topic*.

  If the *notification* of the `SubscriptionEnd` message is supported, the *notification broker* MUST support the persistence of a *subscription*.

A *notification consumer* interested in the `SubscriptionEnd` message MUST be subscribed to the *notification producer* for the *topic* indicated in the `SubscriptionEndTopic` element.

The default is to NOT notify any unexpected subscription termination.

For further details and the XML schema definition (XSD) refer to the Annex 1.

No other special rules or limitations are defined other than those declared in the [OASIS WS-BaseNotification, 2006] and [OASIS WS-BrokeredNotification, 2006] specifications.

2.4.1.2 Data types

See the specification in Section 4.2 of [OASIS WS-BaseNotification, 2006]. See also Annex 1 of this document.

2.4.1.3 Inputs

See the specification in Section 4.2 of [OASIS WS-BaseNotification, 2006].

2.4.1.4 Outputs

See the specification in Section 4.2 of [OASIS WS-BaseNotification, 2006].

2.4.1.5 Errors

See the specification in Section 4.2 of [OASIS WS-BaseNotification, 2006].

2.4.2 Operation "Renew"

This is a request-response operation initiated from the *subscriber* that sends a `Renew` message to a *notification broker* acting as *subscription manager*. In response, the *notification broker* will change the expiration time of the *subscription*.

If the *notification broker* supports the `SubscriptionManager` interface, it MUST implement this operation as defined in Section 6.1.1 of [OASIS WS-BaseNotification, 2006].

Note: the ID of a *subscription* is returned in the response to the `subscribe` request message in the element `/wsnt:SubscribeResponse/wsnt:SubscriptionReference`. To relate the

renew message with the *subscription* to renew this ID must be specified in the renew message header at the element /soap:Header/wsa:To.

It is NOT RECOMMENDED to renew *subscriptions* for an infinite duration.

No other special rules or limitations are defined other than those declared in the [OASIS WS-BaseNotification, 2006] specification.

2.4.2.1 Data types

See the specification in Section 6.1.1 of [OASIS WS-BaseNotification, 2006].

2.4.2.2 Inputs

See the specification in Section 6.1.1 of [OASIS WS-BaseNotification, 2006].

2.4.2.3 Outputs

See the specification in Section 6.1.1 of [OASIS WS-BaseNotification, 2006].

2.4.2.4 Errors

See the specification in Section 6.1.1 of [OASIS WS-BaseNotification, 2006].

2.4.3 Operation "Unsubscribe"

This is a request-response operation initiated from the *subscriber* that sends an UnSubscribe message to the *notification broker* acting as a *subscription manager*. In response, the *notification broker* MUST attempt to destroy the *subscription*.

If the *notification broker* supports the SubscriptionManager interface, it MUST implement this operation as defined in Section 6.1.2 of [OASIS WS-BaseNotification, 2006].

Note: the ID of a *subscription* is returned in the response to the subscribe request message in the element /wsnt:SubscribeResponse/wsnt:SubscriptionReference. To relate the unsubscribe message with the *subscription* to destroy this ID must be specified in the unsubscribe message header of the element /soap:Header/wsa:To.

No other special rules or limitations are defined other than those declared in the [OASIS WS-BaseNotification, 2006] specification.

2.4.3.1 Data types

See the specification in Section 6.1.2 of [OASIS WS-BaseNotification, 2006].

2.4.3.2 Inputs

See the specification in Section 6.1.2 of [OASIS WS-BaseNotification, 2006].

2.4.3.3 Outputs

See the specification in Section 6.1.2 of [OASIS WS-BaseNotification, 2006].

2.4.3.4 Errors

See the specification in Section 6.1.2 of [OASIS WS-BaseNotification, 2006].

2.4.4 Operation "GetCurrentMessage"

The *notification broker* acting as *notification producer* MUST support the GetCurrentMessage message exchange from the NotificationProducer interface [OASIS WS-BaseNotification,

2006]. This is a request-response operation initiated from the *notification consumer* that sends a `getcurrentmessage` message to a *notification broker*. In response, the *notification broker* MAY return the last *notification* published to the given topic. This is a non-destructive read, allowing a newly-subscribed *notification consumer* to get the last *notification* that other *notification consumers* have received.

No other special rules or limitations are defined other than those declared in the [OASIS WS-BaseNotification, 2006] specification.

2.4.4.1 Data types

See the specification in Section 4.3 of [OASIS WS-BaseNotification, 2006].

2.4.4.2 Inputs

See the specification in Section 4.3 of [OASIS WS-BaseNotification, 2006].

2.4.4.3 Outputs

See the specification in Section 4.3 of [OASIS WS-BaseNotification, 2006].

2.4.4.4 Errors

See the specification in Section 4.3 of [OASIS WS-BaseNotification, 2006].

2.4.5 Operation "CreatePullPoint"

This is a request-response operation initiated from the *subscriber* that sends a `CreatePullPoint` message to a *notification broker* acting as *notification producer*. In response the *notification broker* MUST activate a new *pull point*.

The *notification broker* SHOULD implement the interface as defined in Section 5.2 of [OASIS WS-BaseNotification, 2006]. If the operation `CreatePullPoint` is implemented, the related operations `DestroyPullPoint` and `GetMessages` MUST also be implemented.

No other special rules or limitations are defined other than those declared in the [OASIS WS-BaseNotification, 2006] specification.

2.4.5.1 Data types

See the specification in Section 5.2 of [OASIS WS-BaseNotification, 2006].

2.4.5.2 Inputs

See the specification in Section 5.2 of [OASIS WS-BaseNotification, 2006].

2.4.5.3 Outputs

See the specification in Section 5.2 of [OASIS WS-BaseNotification, 2006].

2.4.5.4 Errors

See the specification in Section 5.2 of [OASIS WS-BaseNotification, 2006].

2.4.6 Operation "DestroyPullPoint"

This is a request-response operation initiated from the *subscriber* that sends a `DeletePullPoint` message to the `PullPoint` interface. In response the `PullPoint` MUST attempt to destroy itself.

A *notification broker* acting as a *notification producer* SHOULD implement the interface as defined in Section 5.1.3 of [OASIS WS-BaseNotification, 2006].

Note: the ID of a *pull point* is returned in response to the `CreatePullPoint` message in the element `/wsnt:CreatePullPointResponse/wsnt:PullPoint/wsa:Address`. To relate the `DestroyPullPoint` message with the *pull point* to destroy this ID must be specified in the `DestroyPullPoint` message header at the element `/soap:Header/wsa:To`.

No other special rules or limitations are defined other than those declared in the [OASIS WS-BaseNotification, 2006] specification.

2.4.6.1 Data types

See the specification in Section 5.1.3 of [OASIS WS-BaseNotification, 2006].

2.4.6.2 Inputs

See the specification in Section 5.1.3 of [OASIS WS-BaseNotification, 2006].

2.4.6.3 Outputs

See the specification in Section 5.1.3 of [OASIS WS-BaseNotification, 2006].

2.4.6.4 Errors

See the specification in Section 5.1.3 of [OASIS WS-BaseNotification, 2006].

2.4.7

2.4.8 Operation "GetMessages"

This is a request-response operation initiated from the *notification consumer* that sends a `GetMessages` message to the *pull point*. In response the *notification consumer* will receive the messages accumulated in the *pull point*.

The *notification produce*r SHOULD implement the interface as defined in Section 5.1.2 of [OASIS WS-BaseNotification, 2006].

Note: the ID of a *pull point* is returned in response to the `CreatePullPoint` request message in the element. To relate the `GetMessage` message with the *pull point* this ID must be specified in the `GetMessage` message header at the element `/soap:Header/wsa:To`.

No other special rules or limitations are defined other than those declared in the [OASIS WS-BaseNotification, 2006] specification.

2.4.8.1 Data types

See the specification in Section 5.1.2 of [OASIS WS-BaseNotification, 2006].

2.4.8.2 Inputs

See the specification in Section 5.1.2 of [OASIS WS-BaseNotification, 2006].

2.4.8.3 Outputs

See the specification in Section 5.1.2 of [OASIS WS-BaseNotification, 2006].

2.4.8.4 Errors

See the specification in Section 5.1.2 of [OASIS WS-BaseNotification, 2006].

2.4.9 Operation "RegisterPublisher"

This is a request-response operation initiated from a publisher to confirm its ability to publish on a given *topic* or set of *topics* and to create a publisher registration relation. It is RECOMMENDED that all *publishers* register to the *notification broker* before sending any *notifications*. The *notification broker* MUST be able to receive *notifications* from *publishers* even if they have not registered beforehand.

A publisher sends a `notify` message to the *notification broker* in order to publish a *notification* on a given *topic*.

The *notification broker* MUST implement the interface as defined in Section 6.1 of [OASIS WS-BrokeredNotification, 2006].

A *notification broker* SHOULD indicate through policy assertions or other means the following general behaviour:

- Behaviour in case `PublisherReference` and the `Topic` elements are both missed in the request.
- The default InitialTerminationTime.

No other special rules or limitations are defined other than those declared in the [OASIS WS-BaseNotification, 2006] and [OASIS WS-BrokeredNotification, 2006] specifications.

2.4.9.1 Data types

See the specification in Section 6.1 of [OASIS WS-BrokeredNotification, 2006].

2.4.9.2 Inputs

The input message is based on the specification in Section 6.1 of [OASIS WS-BrokeredNotification, 2006] with the following further constraints:

- /wsn-br:RegisterPublisher/wsn-br:Topic.
- An optional set of `TopicExpressions` that identifies one or more topics. If included, the given publisher is registered to publish only on the set of topics identified by this component. If this is missing the publisher is registered to publish on any topic supported by the *notification broker*.
- /wsn-br:RegisterPublisher/wsn-br:Topic/@Dialect

  A REQUIRED attribute of type unified resource identifier (URI) that identifies the language of the `TopicExpression`. [OASIS WS-Topics, 2006] defines an initial set of standard URIs for `TopicExpressions`. Designers MAY define and use other domain-specific URIs to identify the dialect of the `TopicExpression`.

2.4.9.3 Outputs

See the specification in Section 6.1 of [OASIS WS-BrokeredNotification, 2006].

2.4.9.4 Errors

See the specification in Section 6.1 of [OASIS WS-BrokeredNotification, 2006].

2.4.10 Operation "DestroyRegistration"

This is a request-response operation initiated from publisher to confirm it is willing to destroy a publisher registration.

We recommend that the *notification broker* SHOULD implement the interface as defined in Section 7 of [OASIS WS-BrokeredNotification, 2006].

Note: the ID of a publisher registration is returned in the response to the `RegisterPublisher` request message in the element:

```
/wsn-br:RegisterPublisherResponse/wsn-
br:PublisherRegistrationReference/wsa:Address.
```

To relate the `DestroyRegistration` message with the publisher registration to destroy this ID must be specified in the `DestroyRegistration` message header at the element `/soap:Header/wsa:To`.

No other special rules or limitation other that what defined in the [OASIS WS-BrokeredNotification, 2006] are defined.

2.4.10.1 Data types

See the specification in Section 7.2 of [OASIS WS-BrokeredNotification, 2006].

2.4.10.2 Inputs

See the specification in Section 7.2 of [OASIS WS-BrokeredNotification, 2006].

2.4.10.3 Outputs

See the specification in Section 7.2 of [OASIS WS-BrokeredNotification, 2006].

2.4.10.4 Errors

See the specification in Section 7.2 of [OASIS WS-BrokeredNotification, 2006].

2.4.11 Operation "Notify"

The *notification broker* MUST implement the `NotificationConsumer` interface specified in [OASIS WS-BaseNotification, 2006], MUST support the `Notify` message exchange format and MAY support the raw *notification*. A publisher MUST use this operation to send new notification messages to the *notification broker*. It is RECOMMENDED that the publisher registers beforehand using the `RegisterPublisher` operation.

No other special rules or limitations are defined other than those declared in the [OASIS WS-BrokeredNotification, 2006] specification.

2.4.11.1 Data types

See the specification in Section 3.2 of [OASIS WS-BaseNotification, 2006].

2.4.11.2 Inputs

See the specification in Section 3.2 of [OASIS WS-BaseNotification, 2006.

2.4.11.3 Outputs

See the specification in Section 3.2 of [OASIS WS-BaseNotification, 2006.

2.4.11.4 Errors

See the specification in Section 3.2 of [OASIS WS-BaseNotification, 2006].

3 REFERENCES

[NAC EAPC(AC/322-SC/1)N(2009)0015 (INV), 2009]:
North Atlantic Council/Euro-Atlantic Partnership Council Notice EAPC(AC/322-SC/1)N(2009)0015 (INV), "Core Enterprise Services Framework v1.2", NAC, Brussels, Belgium, 30 Apr 2009 (NATO/EAPC Unclassified).

[NCIA AD 06.05.04.02.E]:
NATO Communications and Information Agency, Agency Directive 06.05.04.02.F, "Service Interface Profile for Publish/Subscribe Services", V. de Sortis, M. Lehmann, R. Fiske, L. Schenkels, D. Gujral, NCIA, The Hague, Netherlands, to be published in 2014, (NATO Unclassified)

[OASIS WS-BaseNotification, 2006]:
Organization for the Advancement of Structured Information Standards (on-line), http://www.oasis-open.org, Web Services Base Notification 1.3 (WS-BaseNotification), at http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf, 1 October 2006, viewed 30 March 2011.

[OASIS WS-BrokeredNotification, 2006]:
Organization for the Advancement of Structured Information Standards (on-line), http://www.oasis-open.org, Web Services Brokered Notification 1.3 (WS-BrokeredNotification), at http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf, 1 October 2006, viewed 30 March 2011.

[OASIS WS-Topics, 2006]:
Organization for the Advancement of Structured Information Standards (on-line), http://www.oasis-open.org, Web Services Topics 1.3 (WS-Topics), at http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf, 1 October 2006, viewed 30 March 2011.

[W3C Xpath, 1999]:
World Wide Consortium (on-line), http://www.w3.org, XML Path Language (XPath) Version 1.0, at http://www.w3.org/TR/xpath, 16 November 1999, viewed 30 March 2011.

4 ABBREVIATIONS

CES	Core Enterprise Services
SIP	Service Interface Profile
SOAP	Simple object access protocol
URI	Unified resource identifier
XML	Extensible mark-up language
XSD	XML schema definition

ANNEX 1 – EXTENSIONS TO OASIS WS-BASENOTIFICATION

A.1 SubscriptionEndTopic element

The `SubscriptionEndTopic` element contains a `TopicExpression` describing the *topic* associated with the `SubscriptionEnd` *notification*.

The structure of the `SubscriptionEndTopic` element is:

```
<wsn-nato:SubscriptionEndTopic
  Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpression/Simple"
  xmlns:wsn-nato="http://nc3a.nato.int/ces/wsn/extension"
  xmlns:tns="http://nc3a.nato.int/ces/wsn/topic" >
        tns:SubscriptionRemoved
</wsn-nato:SubscriptionEndTopic>
```

It contains the following elements:

- `/wsn-nato:SubscriptionEndTopic`

  A `TopicExpression` to be bound to the `SubscriptionEnd` *notification*.

- `/wsn-nato:SubscriptionEndTopic/@Dialect`

  The dialect used in the `TopicExpression`.

A.2 SubscriptionEnd message

In order to inform the *notification consumer* about an unexpected subscription termination, the `SubscriptionEnd` message MUST be created and distributed to subscribed *notification consumers*.

The structure of the `SubscriptionEnd` element is:

```
<wsn-nato:SubscriptionEnd>
  <wsn-nato:SubscriptionReference>
     wsa:EndpointReferenceType
  </wsn-nato:SubscriptionReference>
  {any}*
</wsn-nato:SubscriptionEnd>
```

It contains the following elements:

- `/wsn-nato:SubscriptionEnd`

  An element containing the information about the terminated *subscription*.

- `/wsn-nato:SubscriptionEnd/wsn-nato:SubscriptionReference`

  A reference to the terminated *subscription* as returned from the `Subscribe` operation.

An example of a simple object access protocol (SOAP)-encoded notified `SubscriptionEnd` message is shown below.

```
<s:Envelope ... >
  <s:Header>
     <wsa:Action>
        http://docs.oasis-open.org/wsn/bw-2/NotificationConsumer/Notify
     </wsa:Action>
     ...
  </s:Header>
  <s:Body>
     <wsnt:Notify>
        <wsnt:NotificationMessage>
           <wsnt:SubscriptionReference>
              <wsa:Address>
                   http://www.example.org/SubscriptionManager
              </wsa:Address>
           </wsnt:SubscriptionReference>
           <wsnt:Topic Dialect=
              "http://docs.oasis-open.org/wsn/t-1/TopicExpression/Simple">
              npex:SomeTopic
           </wsnt:Topic>
           <wsnt:Message>
              <wsn-nato:SubscriptionEnd>
                   <wsn-nato:SubscriptionReference>
                        URN:xxx.xxx.xxx
                   </wsn-nato:SubscriptionReference>
                   <reason>
                        Endpoint not reachable
                   </reason>
                   </wsn-nato:SubscriptionEnd>
           </wsnt:Message>
        <wsnt:NotificationMessage>
     </wsnt:Notify>
  </s:Body>
</s:Envelope>
```

A.3 Extention XML Schema

The normative schema for the NATO extension to the WS-Notification specification is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:wsa="http://www.w3.org/2005/08/addressing"
            xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
            xmlns:wsn-nato="http://nc3a.nato.int/ces/wsn/extension"
            targetNamespace=http://nc3a.nato.int/ces/wsn/extension
            elementFormDefault="qualified"
            attributeFormDefault="unqualified">

      <!--======================== Imports  ========================-->
      <xsd:import namespace="http://docs.oasis-open.org/wsn/b-2"
schemaLocation="http://docs.oasis-open.org/wsn/b-2.xsd"/>
      <xsd:import namespace="http://www.w3.org/2005/08/addressing"
schemaLocation="http://www.w3.org/2005/08/addressing/ws-addr.xsd"/>

      <!--======================== Types ============================-->
      <xsd:element name="SubscriptionEnd">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element name="SubscriptionReference"
type="wsa:EndpointReferenceType"/>
                        <xsd:any namespace="##other"
processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
                  </xsd:sequence>
            </xsd:complexType>
      </xsd:element>

      <!--====================== SubscriptionTypes ==================-->
      <xsd:element name="SubscriptionEndTopic"
type="wsnt:TopicExpressionType"/>

</xsd:schema>
```