BizTalk Factory Developer’s Guide

BizTalk Factory is a Microsoft BizTalk Server® enabler framework, like many others out there, be it [BizTalk Server Pipeline Component Wizard](https://btsplcw.codeplex.com/), [BizTalk Software Factory](https://bsf.codeplex.com/) (not to be confused with [BizTalk Factory](https://biztalkfactory.codeplex.com/) itself), [Eliasen's BizTalk Pipeline Components](https://eebiztalkpipelinecom.codeplex.com/), and so on. BizTalk Factory however takes a very unique and distinctive approach in that, instead of being designer/surface or wizard based, it strives to be code centric; wherever possible, BizTalk Factory will try to offer a C# fluent API to replace the designer surface.

# Generic Type Safe API to Read, Promote, and Write Message Context Properties

There is some amount of idiosyncrasies coming with Microsoft BizTalk Server®, and reading, promoting, and writing context properties certainly come with its share of oddities.

First, accessing the context property must be done in two very distinct ways depending on whether the property needs to be accessed from an IBaseMessage or an XLANGMessage message object instance. The IBaseMessage API is even more awkward as one must most often instantiate some sort of context property descriptor object in order to get a QName that will allow one to ultimately get both the context property’s local and namespace names to pass to the API method.

Next, the value returned is always of type object and it must always be casted to the exact concrete type of the property.

Finally, how to delete a property value from the message context as none of the IBaseMessage and XLANGMessage API expose a delete method. Does one have to assign to the property a null value or an empty string? Does this work the same way with both IBaseMessage and XLANGMessage APIs?

IBaseMessage Sample

XLANGMessage Sample

Would it not be simpler if there was a unique, regular, and type-safe way of accessing message context properties? Come Be.Stateless.BizTalk.ContextProperties.MessageContextProperty<T, TR> and Be.Stateless.BizTalk.Message.Extensions.BaseMessage classes to the rescue! With the help of these two classes, and provided the property to access has been defined in a property schema as a Microsoft.XLANGs.BaseTypes.MessageContextPropertyBase-derived property, one now can simply write the following code.

Granted, this new API comes with a price: one has to define the context properties in one property schema, even though the original IBaseMessage API would not require it. All one needed to access a context property from an IBaseMessage was a couple of namespace and local name strings. This new requirement however comes with a huge benefit that far outweighs its drawback. Magic strings never have to be used any more, and namespace can never get out of sync between the IBaseMessage and XLANGMessage APIs.

# BizTalk Factory Context Property Schemas

BizTalk Factory comes with the following context property schemas, all packaged into the Be.Stateless.BizTalk.Schemas assembly:

* Be.Stateless.BizTalk.Schemas.BizTalkFactory.Properties, whose XML target namespace is urn:schemas.stateless.be:biztalk:properties:system:2012:04,
* Be.Stateless.BizTalk.Schemas.Tracking.Properties, whose XML target namespace is urn:schemas.stateless.be:biztalk:properties:tracking:2012:04,
* BTS.UnexposedSystemProperties, whose XML target namespace is <http://schemas.microsoft.com/BizTalk/2003/system-properties>,
* EDI.UnexposedEdiProperties, whose XML target namespace is http://schemas.microsoft.com/Edi/PropertySchema,

# BizTalk Factory Activity Tracking

Activity Model, ActivityTrackerComponent

# BizTalk Factory Batching Subsystem

Microsoft BizTalk Server® comes with a strong de-batching experience thanks to its disassembler pipeline components —that can as well be run within orchestrations;— de-batching is therefore easy and performant. Deceivingly, the Microsoft BizTalk Server® batching experience is very weak. There is no fully functional batching subsystem offered out of the box but only building blocks that are hard to put in place efficiently and correctly, i.e. without affecting the scalability and throughput of the platform. Indeed, most of the known and documented batching/aggregator patterns are based on orchestrations and exhibit either one or all of the following limitations.

* Batching orchestration instances are usually designed to wait for either a new part message to aggregate or a control message to trigger the release of the batch and all its parts accumulated so far. There is an inherent race condition with this design, which manifests itself when both a new part message and a release control message reach the orchestration at the same time. Most of the orchestration designs will fall short in this situation and lead to the following operational issue: The [batching orchestration] instance completed without consuming all of its messages. The instance and its unconsumed messages have been suspended. Fixing the design of the batching orchestration to accommodate for this race condition can only be done, if at all possible, at the expense of an unjustified excessive increase in complexity.
* Batching orchestration instances see their performance gradually decrease as the number of accumulated parts increases. When releasing a batch that is made of 1000 parts or more, it is not unusual that the orchestration takes over 10 minutes to process, that is to say, to construct and send out the envelope message. As one can guess, there certainly is a linear correlation between the orchestration processing time and the number of aggregated parts. Even worse, the longer it takes to process the release of a batch, the more badly designed orchestrations will be exposed to the previously mentioned race condition issue.
* One cannot examine or edit the content of a batch under construction, i.e. the parts that have been accumulated so far, unless at the expense of a relatively complex development. The message parts, which constitute the state of the batching orchestration, are indeed serialized to the message box, which is an opaque repository. Consequently, any serialized orchestration state, and a fortiori its accumulated message parts, cannot be manually edited. One has therefore no other option than to create custom code to release the batch and process it through an alternate process that will allow its parts to be edited before being accumulated all over again.

BizTalk Factory departs from an orchestration-based batching solution and instead offers a Microsoft SQL Server® based batching subsystem that directly addresses all of the latter issues. The operational issues that arise consequently to the occurrences of the race condition with the orchestration-based design have gone. Though the race condition still subsists with the BizTalk Factory batching subsystem, it has been addressed at the database level and no more can the release of a batch ends up being suspended. The batch release process, i.e. the construction of the message envelope, is performed by the database in nearly constant time, even for a very large number of accumulated parts. At any time, one can have a look at or edit the content of a batch or its parts through simple T-SQL statements.

## Batching Subsystem Overview

From the surface, the BizTalk Factory batching subsystem is made up of a series of send ports and receive location whose purpose is either to accumulate message parts out of the BizTalk message box, or queue the release of batches upon reception of control messages, or inject the content of batches just released into the BizTalk message box. Before looking into the technical details on how these concrete BizTalk artifacts interact and operate to deliver the BizTalk Factory batching subsystem, there is a couple of distinctive features that need to be grasped.

### Envelope Schemas and Partitions

Without surprise, the batching subsystem of BizTalk Factory is intrinsically related to an envelope schema —technically an EnvelopeSpecName— but it also intrinsically relies on the notion of partition. Conceptually, batch partitioning is some sort of constrain that ensures that, when releasing a batch and its parts, only the parts belonging to the same partition will be aggregated together within one envelope message. Technically, a batch must therefore be denoted by both an envelope schema and a partition.

Notice that even though envelope schemas must be registered to enable the batching subsystem to start accumulating parts, partitions, which are just string labels, are totally dynamic and do not require any prior registration. Moreover, partitions are also optional and parts do not have to belong to any partition to be accumulated. Technically though, when a part being accumulated has no belonging partition, it will be associated to the default 0 (zero) partition.

### Batch Releases and Batch Release Policies

Batches can be released either upon the reception of a control message —that is an instance of the Xml.Batch.Release message schema/type— or according to some dynamically evaluated release policies. These release policies, which can be configured at the batch level whether partitioned or not, are based on the following parameters:

* Enabled: For a given batch, designated by an EnvelopeSpecName and Partition couple, this parameter denotes whether the release policy for the batch is enabled or not. Disabling a release policy also prevents batches from being released via control messages.
* Partition: Denotes the specific partition of the related EnvelopeSpecName for which this release policy applies. If no partition is explicitly specified, it defaults to the 0-partition.
* ReleaseOnIdleTimeOut: For a given batch, designated by an EnvelopeSpecName and Partition couple, this parameter denotes the maximum amount of time that can elapse, since the last accumulated batch item or part, before the batch is automatically released.
* ReleaseOnElapsedTimeOut: For a given batch, designated by an EnvelopeSpecName and Partition couple, this parameter denotes the maximum amount of time that can elapse, since the very first accumulated batch item or part, before the batch is automatically released. It also ensures that any batch will eventually be released independently of the sliding nature of the ReleaseOnIdleTimeOut criterion.
* ReleaseOnItemCount: For a given batch, designated by an EnvelopeSpecName and Partition couple, this parameter denotes the minimum number of batch items or parts that can be accumulated before the batch is automatically released.
* EnforceItemCountLimitOnRelease: For a given batch, designated by an EnvelopeSpecName and Partition couple, this parameter denotes whether the ReleaseOnItemCount parameter is also used to enforce a maximum size limit on the number of batch items or parts that can be released together in a single envelope message. If the EnforceItemCountLimitOnRelease criterion is disabled, then all the batch items or parts that have been accumulated so far for a given batch will be released altogether.

Among these parameters, only the Enabled one is mandatory. Consequently if none of the optional parameters have been configured, batches will never be released automatically and only control messages would be able to release them.

Moreover, neither the reception of a control message nor the satisfaction of some release policy will trigger the immediate release of a batch; they will rather schedule the imminent release of batch. The actual release will indeed happen when BizTalk Server polls for the batches ready to be released, at which time, all the available batches will be released in a row, one after the other.

## Batching Subsystem Design

The following diagram provides a general overview of how the BizTalk Factory batching subsystem is materialized within Microsoft BizTalk Server®. At the Microsoft BizTalk Server’s perimeter, there are a couple of static one-way send ports and a single one-way receive location.



Figure 1 — Batching Subsystem Design Overview

* BizTalk.Factory.SP1.Batch.AddPart.WCF-SQL.XML, which is meant to aggregate parts for a given envelope. This send port subscribes to any message having a BizTalkFactory.EnvelopeSpecName property promoted in context and will store its XML payload in database as a part of the corresponding batch, which is denoted by the BizTalkFactory.EnvelopeSpecName and BizTalkFactory.EnvelopePartition context properties, should the message have both, or denoted by the BizTalkFactory.EnvelopeSpecName context property and the default 0-partition, should the message have no given partition in context.  
  Notice that only the BizTalkFactory.EnvelopeSpecName context property is part of the send port’s filter subscription, and consequently only it must be promoted in context; the BizTalkFactory.EnvelopePartition context property, should it be present, simply has to be written in context.
* BizTalk.Factory.SP1.Batch.QueueControlledRelease.WCF-SQL.XML, which is meant to schedule a batch for imminent release upon reception of a control message. This send port subscribes to any message being an instance of the Xml.Batch.Release schema/type.
* BizTalk.Factory.RL1.Batch.Release.WCF-SQL.XML, which is actually responsible of releasing a batch, that is to say its envelope and all its parts. This receive location regularly polls the database for the next available batch to release, whether releasable because a release policy has become satisfied at poll time or because a release control message has been received earlier. Recall that all the available batches will be released in a row, but one after the other to avoid saturating the receive location and the message box with a potentially large set of potentially big envelope messages arriving all at once.  
  Notice that when an envelope message is published to the message box, its partition will be promoted, via the BizTalkFactory.EnvelopePartition context property, on top of its message type, via the BTS.MessageType context property, as usual.

Caution. All the CLR property names and message type names that have been used in the preceding description about the BizTalk Server send ports and receive location were truncated for the sake of readability. In reality, these names are all prefixed by Be.Stateless.BizTalk.Schemas. This way, the complete and correct CLR name for the BizTalkFactory.EnvelopeSpecName property is Be.Stateless.BizTalk.Schemas.BizTalkFactory.EnvelopeSpecName. Similarly, the complete and correct name for the Xml.Batch.Release message type is Be.Stateless.BizTalk.Schemas.Xml.Batch.Release.

Caution. BizTalkFactory.EnvelopeSpecName is not to be confused with BTS.EnvelopeSpecName. BizTalk Factory has deliberately chosen to use an equivalent property but in a different namespace to totally insulate its batching subsystem from any batch-related feature built in Microsoft BizTalk Server®.

### Release Control Message

The release control message is used to schedule an imminent release of a given batch and has the following structure, where the EnvelopeSpecName is mandatory and the Partition is optional. Its CLR full name is Be.Stateless.BizTalk.Schemas.Xml.Batch.Release, while its XML message type is urn:schemas.stateless.be:biztalk:batch:2012:12#ReleaseBatch.



Figure 2 — Batch Release Control Message Structure

Recall that a batch is unequivocally denoted by an envelope schema —technically an EnvelopeSpecName— and a Partition. As a consequence, failing to specify a partition in the release control message will only release the default 0-partition of a batch, and no other partition should there be more than one.

However, this message also accepts the \* (any) wildcard instead of either the EnvelopeSpecName or the Partition elements, or both. One could therefore easily release all the partitions of a given envelope schema (the message would specify a given EnvelopeSpecName but a \* Partition), or only a given partition irrespectively of the envelope schema (the message would specify a \* EnvelopeSpecName and a given Partition), or all the batches having been accumulated so far, whatever the envelope schema and partition (the message would specify a \* EnvelopeSpecName and Partition).

### Batch Content Message

As depicted on Figure 1 — Batching Subsystem Design Overview, the BizTalk.Factory.RL1.Batch.Release.WCF-SQL.XML receive location relies on a T-SQL stored procedure, usp\_batch\_ReleaseNextBatch, to receive a message with the content of a given batch. Technically, this stored procedure is used to build the batch content message, whose structure is depicted on Figure 3 — Batch Content Message Structure. Its CLR full name is Be.Stateless.BizTalk.Schemas.Xml.Batch.Content, while its XML message type is urn:schemas.stateless.be:biztalk:batch:2012:12#BatchContent. Only the EnvelopeSpecName, Partition, and Parts elements should be familiar; the ProcessActivityId and MessagingStepActivityIds elements will be described in the subsequent Batching Subsystem Activity Monitoring section.

Notice that this batch content message is not the expected envelope message as it is clearly not an instance document of the envelope schema referenced by the EnvelopeSpecName element. This batch content message is indeed a private message —i.e. internal to the BizTalk Factory batching subsystem— that has still to undergo some transformation to finally correspond to the expected envelope schema. This is the topic of the following section, Envelope Builder Component.



Figure 3 — Batch Content Message Structure

### Envelope Builder Component

The BizTalk.Factory.RL1.Batch.Release.WCF-SQL.XML receive location is relying on the Be.Stateless.BizTalk.Pipelines.BatchReceive receive pipeline, which itself embodies a Be.Stateless.BizTalk.Component.EnvelopeBuilderComponent pipeline component. The purpose of this pipeline component it to apply a generic XSLT transform on the batch content message returned from the database in order to convert it into an instance document of the expected envelope schema.

At first sight, applying a generic XSLT might seem impossible as there will certainly be a variety of envelope schemas, probably very different from one another. In order to do so, the EnvelopeBuilderComponent relies on composite multi-part message similar to the XML excerpt that follows —for which there is no corresponding message schema deployed into Microsoft BizTalk Server®.

<agg:Root xmlns:agg="http://schemas.microsoft.com/BizTalk/2003/aggschema">

<agg:InputMessagePart\_0>

<ns0:Envelope xmlns:ns0="urn:schemas.stateless.be:biztalk:envelope:2013:07>

**<ns:parts-here xmlns:ns="urn:schemas.stateless.be:biztalk:batch:2012:12" />**

</ns0:Envelope>

</agg:InputMessagePart\_0>

<agg:InputMessagePart\_1>

<ns:BatchContent xmlns:ns="urn:schemas.stateless.be:biztalk:batch:2012:12">

<ns:EnvelopeSpecName>...</ns:EnvelopeSpecName>

<ns:Partition>...</ns:Partition>

<ns:Parts>

...

</ns:Parts>

</ns:BatchContent>

</agg:InputMessagePart\_1>

</agg:Root>

This composite multi-part XML exhibits a couple of key characteristics that will allow a generic XSLT to be applied, no matter what is the expected envelope schema. Primarily, the agg:InputMessagePart\_0 element will always contain a dummy message that is an instance of the expected envelope schema and the agg:InputMessagePart\_1 element will always contain the batch content message as returned from the database. Furthermore, the dummy envelope instance message will always contain a placeholder —i.e. the parts-here element at the envelope schema’s BodyXPath expression— that has to be replaced with the verbatim content of the inner XML of the agg:InputMessagePart\_1/ns:BatchContent/ns:Parts element.

### EnvelopeMapSpecName Envelope Schema Annotation

At times, the envelope message just built and submitted to the message box will immediately undergo another XSLT transformation on the part of its subscriber. To avoid two transformations in a row, it is possible to instruct the EnvelopeBuilderComponent pipeline component to apply a custom XSLT stylesheet on the composite multi-part message. One simply has to annotate the target envelope schema with an EnvelopeMapSpecName instruction denoting the fully qualified name of the custom XSLT transform to apply, as in the following excerpt.

<xs:schema targetNamespace="..." xmlns:xs="http://www.w3.org/2001/XMLSchema"

xmlns:san="urn:schemas.stateless.be:biztalk:annotations:2013:01">

<xs:element name='Envelope'>

<xs:annotation>

<xs:appinfo>

**<san:EnvelopeMapSpecName>**

Be.Stateless.BizTalk.Unit.Transform.IdentityTransform,

Be.Stateless.BizTalk.Unit, Version=1.0.0.0, Culture=neutral,

PublicKeyToken=3707daa0b119fc14

**</san:EnvelopeMapSpecName>**

...

</xs:appinfo>

</xs:annotation>

...

</xs:element>

</xs:schema>

Note. To build the composite multi-part message, the pipeline component relies on an instance of the Be.Stateless.BizTalk.Streaming.CompositeStream class whose content will be lazily —i.e. on demand while being read— constructed without any memory overhead.

### Envelope Message Publishing

Ultimately, the envelope message that is received by the BizTalk.Factory.RL1.Batch.Release.WCF-SQL.XML receive location and comes out of the Be.Stateless.BizTalk.Pipelines.BatchReceive receive pipeline is published to the message box. To better allow the interested parties to subscribe to this envelope message, its partition has been promoted into context, via the BizTalkFactory.EnvelopePartition context property, on top of the message type, via the BTS.MessageType context property, as usual.

### Data Model Overview

Figure 4 — Batching Subsystem Data Model illustrates the main database objects underlying the BizTalk Factory batching subsystem. Unsurprisingly, the envelope schemas, represented by the batch\_Envelopes table, are central in the data model and every other feature is related to them. The batch items, or parts, that are stored in the batch\_Parts table, have to unequivocally relate to an envelope, which ultimately stands for an envelope schema, i.e. an EnvelopeSpecName. Similarly, the release policy definitions, represented by the batch\_ReleasePolicyDefinitions table, are also individually linked to an envelope schema. Even the queueing/scheduling of batch releases consequent to the reception of control messages, represented by the batch\_QueuedControlledReleases table, are linked to an envelope schema.

Figure 4 also exhibits that the partition defaults to 0 and is thus a truly optional feature. Notice however that partition is a free text string in all of the batch\_Parts, batch\_ReleasePolicyDefinitions, and batch\_QueuedControlledReleases tables; it is therefore the developer’s responsibility to guarantee that partition names will be consistent across all the three tables.

Looking back at Figure 1 — Batching Subsystem Design Overview, one can see that the Microsoft BizTalk Server® artifacts are not directly interacting with the database tables, but only with some view and stored procedures. These stored procedures —namely usp\_batch\_AddPart, usp\_batch\_QueueControlledRelease, and usp\_batch\_ReleaseNextBatch,— which are not depicted on the diagram, are rather self-explanatory. The vw\_batch\_ReleasePolicies and vw\_batch\_NextAvailableBatch views, however, deserve some explanation.

The vw\_batch\_ReleasePolicies view evaluates the release policy definitions and shows their results, i.e. the list of every batch —denoted by an EnvelopeSpecName and Partition— that satisfies its policy definition predicates. Notice that for the sake of scalability and throughput, and at the expense of consistency, this view does not acquire any database locks.

The vw\_batch\_NextAvailableBatch view, together with the usp\_batch\_ReleaseNextBatch stored procedure, is used by the polling receive location to receive batches and publish envelope messages to the message box. This view integrates both the results of the previously mentioned vw\_batch\_ReleasePolicies view and the data of the batch\_QueuedControlledReleases table —i.e. the batches scheduled for release consequent to the reception of control messages— to determine what will be the next batch to release, or alternatively, what will be the next envelope message to publish. This view and its partnering stored procedure have been designed for consistency and correctness; they do consequently acquire database locks, at the expense of scalability and throughput, though great care has been taken to limit the impact as much as possible.



Figure 4 — Batching Subsystem Data Model

Caution. As the vw\_batch\_NextAvailableBatch view acquires database locks, it should not be used interactively, not even for operational or administrative purposes. To circumvent this shortcoming, should an operator agent still be willing to peek into the inner workings of the batching subsystem, the vw\_batch\_AvailableBatches view has been created. This view offers a similar and even richer functionality than the vw\_batch\_NextAvailableBatch view —it lists all the batches that are ready to be released and not only the first one,— but without acquiring any lock and, of course, at the expense of consistency.

### Batch Registration

Last but not least, in order to be able to accumulate batch items for a given batch, one has first to register it. A batch, denoted by an EnvelopeSpecName and Partition, can be registered via the usp\_batch\_Register stored procedure, which should typically be called at deployment time. This stored procedure can moreover be used to configure the release policy of the batch being registered, as depicted by its signature that follows.

PROCEDURE [dbo].[usp\_batch\_Register]

@envelopeSpecName nvarchar(256),

@partition nvarchar(128) = '0',

@enabled bit,

@releaseOnElapsedTimeOut int = null,

@releaseOnIdleTimeOut int = null,

@releaseOnItemCount int = null,

@enforceItemCountLimitOnRelease bit = 0

AS

Reversely, there is an usp\_batch\_Unregister stored procedure that should be called at un-deployment time so as to deregister a batch and its release policy.

## Batching Subsystem Activity Monitoring

# BizTalk Factory Claim Check

# BizTalk Factory Pipeline Components

# BizTalk Factory Micro Pipeline

# BizTalk Factory Pipeline DSL

# BizTalk Factory Rule DSL

# BizTalk Factory Binding DSL

# BizTalk Factory Service Model

# BizTalk Factory Unit

# BizTalk Factory Monitoring Web Site

# BizTalk Factory Goodies

Message extension methods to get direction, to compress, to base64, etc…

CompositeStream, CompositeXmlReader, FileStreamTransacted, XmlBuilderReader (from IEnumerable to IXmlReader), XSLT message context injection, etc.