<https://osmanhbiztalkblog.wordpress.com/2015/06/22/ejemplo-de-itinerario-para-un-proceso-request-response-con-esb-toolkit-2-3/>

# Diseño de un servicio Request-Response con ESB ToolKit 2.3

[junio 22, 2015](https://osmanhbiztalkblog.wordpress.com/2015/06/22/ejemplo-de-itinerario-para-un-proceso-request-response-con-esb-toolkit-2-3/) [ohawarip](https://osmanhbiztalkblog.wordpress.com/author/ohawarip/) [ESB](https://osmanhbiztalkblog.wordpress.com/category/esb/)

En este artículo trataremos de definir los pasos que se deben seguir para implementar una solución en BizTalk 2013R2 utilizando el Framework **ESB Toolkit 2.3**

El ejemplo que implementaremos es bastante sencillo, ya que consiste en la implementación de un itinerario básico que permite la integración con un servicio web.

El itinerario tiene cuatro componentes fundamentales:

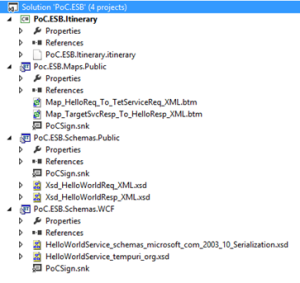
* **Recepción de la petición**: la petición se recibe en el itinerario en un formato de mensaje público manejado por los clientes. Esta petición una vez recibida es transformada en un mensaje que sea entendible por el sistema destino al que deseamos invocar. De cara a la reutilización, la configuración de las transformaciones se delega en un conjunto de reglas de negocio.
* **Enrutamiento**: este proceso consiste en determinar el sistema de destino al que se enviará la petición. En nuestro caso, para hacer el itinerario más potente y reutilizable, la configuración de este proceso se delega en un conjunto reglas de negocio.
* **Consumo del servicio**: una vez que tenemos identificado el destino, el siguiente paso, y más importante consiste en consumir el servicio destino. Para ello se establece dinámicamente la configuración del sistema destino al que enviar la petición. Toda esta parametrización se delega en reglas de negocio.
* **Generación de la respuesta**: cuando se recibe la respuesta desde el sistema destino, este mensaje es transformado previamente en un mensaje entendible por el cliente en el paso previo a devolver la respuesta. Del mismo modo que en los casos anteriores, toda esta configuración se delega en reglas de negocio.

El desarrollo de aplicaciones basadas en el ESB Toolkit, en general, no difiere casi en nada con respecto a la creación de una aplicación BizTalk estándar. El primer paso que se debe seguir consiste en la creación de la solución BizTalk que contenga todos los artefactos requeridos para el proceso. En nuestro caso, la aplicación BizTalk contiene lo siguiente:

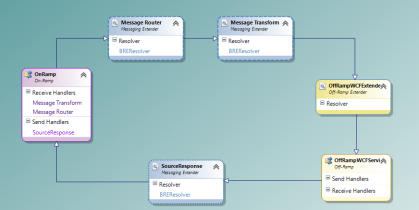
* **Esquemas de mensajería públicos**: se deben definir los esquemas de mensajería públicos creados para el consumo del servicio por los clientes.
* **Esquemas de mensajería destino**: es necesario definir los esquemas de mensajería que se utilizarán para la comunicación con el sistema destino.
* **Transformaciones**: es necesario definir las transformaciones de mensajería que se deben utilizar para adaptar la mensajería pública a la esperada por el servicio destino y viceversa.

Adicionalmente y como parte del desarrollo mediante el uso del ESB, se crea también un **itinerario** que no es más que una representación gráfica del workflow del proceso a implementar. Así pues, salvo este componente, el desarrollo es similar al de cualquier aplicación BizTalk estándar.

En nuestro caso, la solución que hemos construído es la siguiente:

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/sln.png)

Y el itinerario que define el flujo del proceso qued ade la siguiente forma:

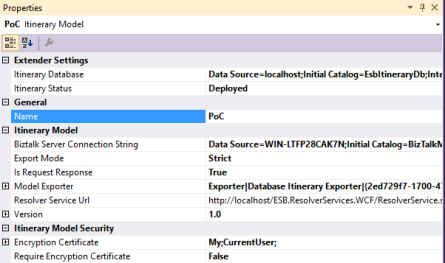
[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/itin.png)

En el diseño del itinerario, cabe destacar lo siguiente:

* El itinerario es de tipo **Request-Response**, dado que va a ser consumido de forma síncrona mediante la publicación de un servicio web.
* El **On-Ramp** representa la recepción de la petición desde el cliente externo y que inicia el itinerario. Adicionalmente representa también el punto por el cual se devuelve la respuesta al cliente.
* El **Off-Ramp** representa la invocación al servicio de destino.
* El itinerario contiene varios **Messaging Extender**. Estos componentes representan dependiendo la su tipología o bien transformaciones o bien routing de mensajería.
* El **Off-Ramp Extender**se utiliza para establecer la información necesaria para la invocación del sistema externo.

Profundizando en mayor detalle en el itinerario, podemos visualizar cada una de las propiedades de los artefactos y del propio itinerario en sí.

Con respecto a las propiedades del itinerario, tenemos lo siguiente:

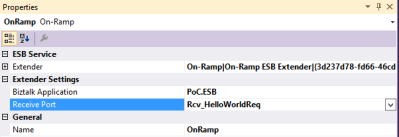
[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/itinprop.png)

En nuestro caso, indicamos que el itinerario es exportado directamente a la BD de itinerarios. Existe la posibilidad de exportarlo a XML y luego es posible publicarlo en la BD mediante un comando.

Otra propiedad interesante es la posibilidad de utilizar un certificado para cifrar el contenido del itinerario, ya que puede haber información sensible. Dado que esto es una prueba, no es necesario.

Por último, cabe resaltar la propiedad **Is Request Response**. Esta propiedad es importante ya que indica si el itinerario va a devolver una respuesta al cliente que lo invoca en el OnRamp o no. En nuestro caso sí que lo es ya que se consume mediante un servicio web.

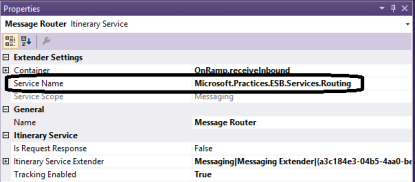
En cuanto a las propiedades del **On-Ramp**, tenemos lo siguiente:

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/onrampprop.png)

Cabe resaltar fundamentalmente las propiedades **BizTalk Application** y **Receive Port**. Básicamente se indica a través de que puerto de la aplicación BizTalk indicada se reciben las peticiones en el itinerario.

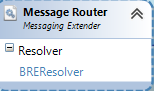
Los artefactos de tipo **Messaging Extender**, como bien habíamos dicho antes, pueden ser de dos tipos: **Transform** o **Routing**. El artefacto en si es el mismo, la única diferencia es el tipo de servicio que implementa.

Por ejemplo, la caja Message Router es de tipo Routing y tiene las siguientes propiedades:

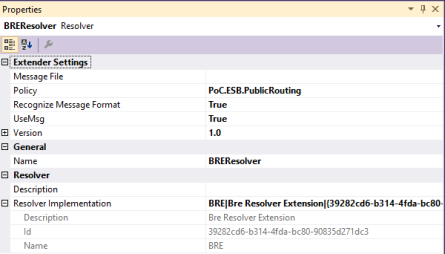
[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/routerprop.png)

Como habíamos indicado, el **Service Name** indica el tipo de artefacto, en este caso routing. Otra propiedad muy importante es el **Container** que básicamente indica en que OnRamp/OffRamp se ejecuta el componente y si se ejecuta en la recepción o en el envío de información. En nuestro caso, se ejecuta en el OnRamp en la recepción del mensaje de petición **On-Ramp** -> **receiveInbound**.

Si nos fijamos en la caja del Message Router en el itinerario, vemos lo siguiente:

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/routerprop1.png)

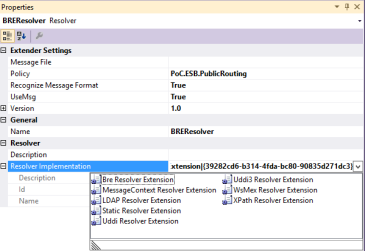
Tiene un componente llamado **BREResolver** dentro de la lista de Resolvers. Estos componentes son los que dotan de la funcionalidad real a los Messaging Extender. En nuestro itinerario, este componente tiene la siguiente configuración:

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/breresolver.png)

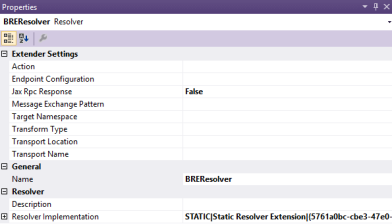
Sin duda, la propiedad más importante y la que antes se debe fijar es el tipo de implementación del resolver. En nuestro caso, hemos seleccionado **BRE**, dado que la configuración del routing la llevaremos a cabo de forma dinámica utilizando reglas de negocio BRE.

Una vez seleccionada la implementación, se habilitan el resto de opciones. Nos pide seleccionar la política del motor de reglas a ejecutar y la versión. Adicionalmente se indica si se debe reconocer el mensaje procesado y si se debe utilizar el mensaje.

Como se puede apreciar en la siguiente captura, existen varios tipos de Resolvers:

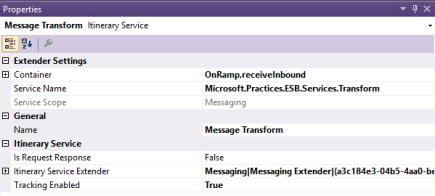
[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/resolvertypes.png)

Siendo el más simple el **Static Resolver**, ya que implica que la configuración es estática en el itinerario, es decir, fija tal y como se muestra a continuación:

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/resolverstatic.png)

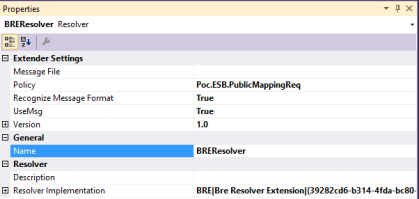
Como se puede apreciar en la imagen, es preciso establecer de forma estática la configuración de enrutado que se utilizará en el itinerario.

Si seleccionamos la caja de Message Transform, podemos ver que el Messaging Extender está configurado utilizando el Service Name de Transform, tal y como se puede ver a continuación:

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/mesassingtransform.png)

La configuración es idéntica a la de la caja de routing, pero cambiando el Service Name. Como se puede apreciar, se ejecutará la transformación al recibir la petición en el **On-Ramp**.

Del mismo modo, esta caja tiene un resolver, con la siguiente configuración:

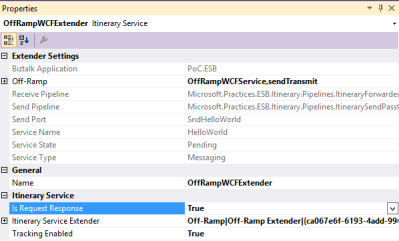
[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/transformresolver.png)

Se trata también de una configuración basada en **BRE**, pero con una política de negocio distinta.

La diferencia entre los Messaging Extender radica en la configuración de los mismos. Los de tipo **Routing**, precisan que se rellene la información necesaria para invocar al servicio de destino. Como se puede ver en las propiedades estáticas, es necesario indicar el T**ransport Location**, **Transport Type**, **Transport Name**, **Endpoint Configuration**, etc…

Sin embargo, para los de tipo **Transform**únicamente requieren que se rellene la propiedad **Transform Type**.

El siguiente artefacto que es preciso mencionar es el **Off-Ramp Extender**. Las propiedades de este componente se representan en la siguiente captura:

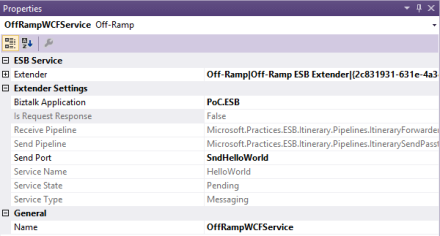
[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/offext.png)

Básicamente es preciso indicar el **Off-Ramp** al que está asociado. Al hacer esto, se rellenará toda la información relativa al **Off-Ramp** que aparece deshabilitada.

Otra porpiedad muy importante es **Is Request Response**. En caso de que la comunicación del **Off-Ramp** sea de tipo **Two-Way**, es necesario establecer esta propiedad a True. En nuestro caso la propiedad está a True dado que invocaremos a un servicio web.

El **OffRamp** extender es obligatorio siempre que se use un **Off-Ramp**.

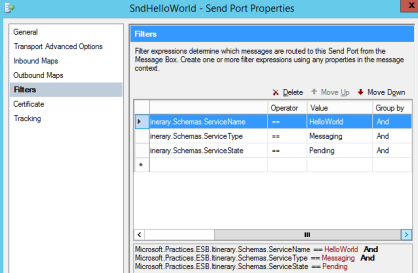
Acto seguido, el siguiente artefacto a detallar es el **Off-Ramp** en sí. Este artefacto representa la comunicación con el sistema externo y tiene la siguiente configuración:

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/offramp.png)

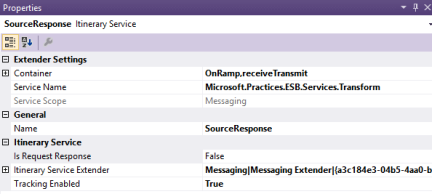
Básicamente se indica la aplicación BizTalk y el puerto de envío **dinámico**que se utilizará para la comunicación dentro del OffRamp. El puerto debe ser dinámico para establecer al configuración.

Si se presta atención la configuración, existen varias propiedades que están deshabilitadas. El motivo es que dichas propiedades se establecen al seleccionar el puerto. Peor hay que tener especial atención con las propiedades **Service Name, Service State y Service Type**. Estas propiedades se deben establecer como filtros en el puerto de la aplicación de BizTalk, o no serán rellenadas y el proceso fallará.

En nuestro caso, el puerto contiene las siguientes propiedades:

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/spfilter.png)

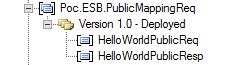
La última caja que contiene el itinerario es la caja denominada **Source Response**. Esta caja es un **Messaging Extender**de tipo **Transform**, pero que a diferencia de los otros, se ejecuta al enviar la respuesta en el **On-Ramp** tal y como se puede apreciar en la propiedad **Container**.

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/sourceresp.png)

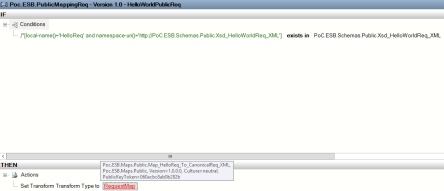
Una vez definido el itinerario, es preciso explicar cómo se implementan las políticas de negocio, que son las que realmente definen la configuración de todo el proceso.

En nuestro ejemplo, tenemos dos políticas de negocio, una que define las transformaciones a aplicar en función de algunas condiciones y otra que define las propiedades de enrutamiento.

La política que define las transformaciones tiene dos reglas, una para la transformación de las peticiones y otra para la transformación de las respuestas:

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/bremap1.png)

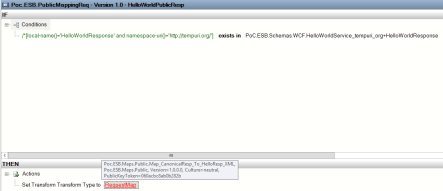
La regla que se ejecuta al recibir la petición, tiene la siguiente estructura:

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/bremap2.png)

Como se puede apreciar, se establece la transformación en el caso de que se cumpla la condición. En nuestro caso la condición es que el servicio sea el de **HelloWorld**.

Es importante resaltar que se utiliza el vocabulario **Set Transform Transform Type**. Este vocabulario pertenece a **ESB.TransformInfo** el cual establece las propiedades de contexto necesarias en el mensaje para que el **Messaging Extender** de tipo **Transform** puede aplicar el mapa indicado.

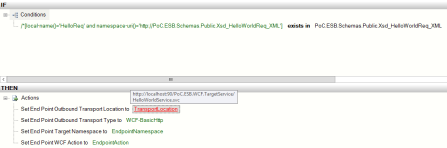
De igual forma, la regla de respuesta es la siguiente:

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/bremapresp.png)

La política que define el enrutamiento tiene una sola regla tal y como se indica a continuación:

[brerout1](https://osmanhbiztalkblog.files.wordpress.com/2015/06/brerout1.png)

Esta regla tiene la siguiente estructura:

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/brerout2.png)

De la misma forma que con la de transformación, se establecen otras propiedades. Estas propiedades pertenecen al vocabulario **ESB.EndPointInfo**. Estas propiedades serán utilizadas por el **Messaging Extender** de **Routing**para enviar la petición al servicio indicado.

El último punto que merece la pena que sea definido es la configuración tanto del puerto de recepción como del puerto de envío en la aplicación de BizTalk.

Cuando diseñamos una aplicación sobre el ESB Toolkit, tenemos dos opciones para la configuración del puerto de entrada. O bien podemos usar uno de los proporcionados por el propio ESB, o bien podemos crear nuestro puerto de entrada en la aplicación.

La diferencia fundamental entre ambos sistemas es que en caso de usar uno de los proporcionados por el ESB, tendremos que trabajar con mensajería no tipada, ya que estos puntos de entrada son genéricos y deben funcionar para cualquier aplicación.

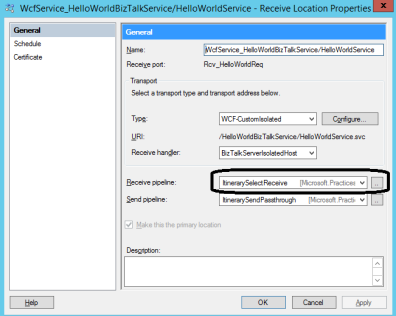
Un ejemplo de petición de un Web Service proporcionado por el ESB sería el siguiente:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <**soap:Envelope** xmlns:soap="<http://www.w3.org/2003/05/soap-envelope>"    xmlns:itin="<http://schemas.microsoft.biztalk.practices.esb.com/itinerary>">    <**soap:Header**>    <**itin:ItineraryDescription**>    <**Name**></**Name**>    <**Version**></**Version**>    <**Guid**></**Guid**>    </**itin:ItineraryDescription**>    </**soap:Header**>    <**soap:Body**>    <**part**></**part**>    </**soap:Body**>    </**soap:Envelope**> |

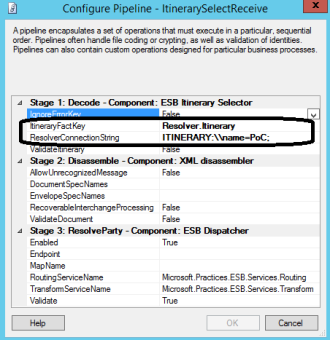
Como se puede apreciar, el payload del mensaje es no tipado y va dentro del nodo part. Esto dificulta el consumo del servicio por parte de los clientes.

En nuestro caso, hemos optado por la otra alternativa, hemos definido un Web Service tipado y hacemos uso de los **Pipelines** proporcionados por el ESB, de forma que indicamos el itinerario que queremos iniciar cuando invoquemos al servicio.

Esta configuración la establecemos en el **Pipeline** de recepción de la ubicación de entrada.

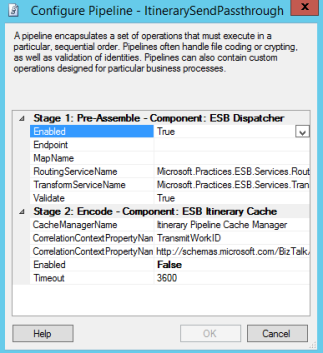
[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/rcloc.png)

Como podemos apreciar, usamos un Pipeline denominado **ItinerarySelectReceive**, el cuál es proporcionado por el propio ESB. Si abrimos la configuración podemos ver lo siguiente:

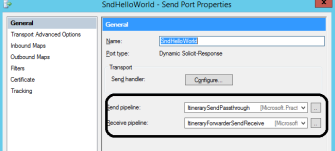
[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/rcloc1.png)

En nuestro caso, indicamos estáticamente que queremos iniciar el itinerario **PoC** siempre que recibamos una petición. Adicionalmente, este itinerario da soporte a Messaging Extender de Routing y Transform. Como al recibir la petición debemos ejecutar un Routing y un Transform en el itinerario, precisamos usar este componente.

Con respecto al Pipeline de Respuesta, utilizamos el Pipeline **ItinerarySendPassthrough**. Este Pipeline es el más básico del ESB y sólo da soporte a Messaging Extender de Routing y Transform. Como a la respuesta aplicamos una transformación, necesitamos utilizarlo.

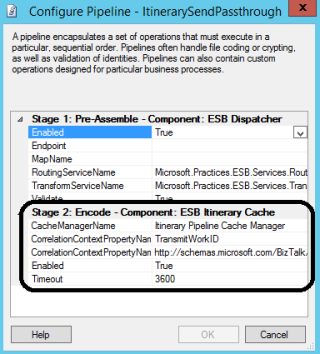
[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/rcloc2.png)

Con respecto al puerto de envío para el consumo del servicio externo, definimos un puerto **dinámico**de tipo **request-response** y lo configuramos de la forma siguiente:

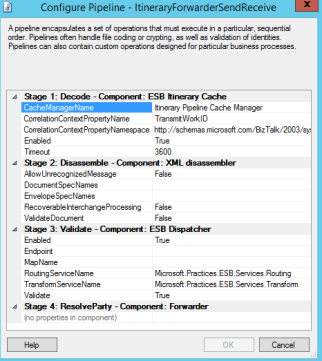
[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/sp1.png)

Como se puede apreciar, utilizamos otros dos Pipelines, los cuales son proporcionados por el ESB.

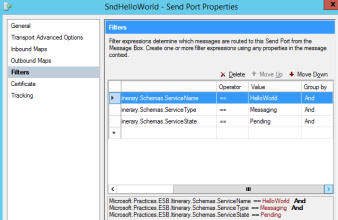
En el caso del envío, utilizamos el Pipeline **ItinerarySendPassthrough**. Este pipeline aparte de permitir los Extenders de Routing y Transform, nos permite establecer las propiedades necesarias de modo que podamos continuar con la ejecución del itinerario al recibir la respuesta:

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/sp2.png)

Con respecto a la recepción, usamos el Pipeline **ItineraryForwarderSendReceive**, el cuál aparte de permitir la ejecución de Messaging Extenders, nos permite continuar con la ejecución del itinerario al recibir la respuesta del servicio web. Adicionalmente tiene un **XML Disassembler** que nos permite inferir el tipo del mensaje recibido de respuesta.

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/sp3.png)

Por último, tal y como indicábamos anteriormente, el puerto de envío contiene los filtros necesarios para poder ser invocado desde el itinerario.

[](https://osmanhbiztalkblog.files.wordpress.com/2015/06/spfilter1.png)

El código asociado a la solución se puede desargar desde el siguiente enlace:

<https://onedrive.live.com/redir?resid=4176A95AD936E072!1066&authkey=!AOq4P3BwrZ4R_e8&ithint=file%2czip>

**Sponsored Content**