Eusko Jaurlaritzaren Informatika Elkartea
Sociedad Informática del Gobierno Vasco

W91D
ENVÍO DE MENSAJERÍA SMS PNS
GUÍA DE DESARROLLO

Fecha: 14/01/2016

Referencia: W91D

EJIE S.A.
Mediterráneo, 3
Tel. 945 01 73 00*
Fax. 945 01 73 01
01010 Vitoria-Gasteiz
Posta-kutxatila / Apartado: 809
01080 Vitoria-Gasteiz
www.ejie.es

Eusko Jaurlaritzaren Informatika Elkartea
Sociedad Informática del Gobierno Vasco

Control de documentación

Título de documento: LATINIA_MANTENIMIENTO_APLICACIONES.DOC

Histórico de versiones

Código: W91

Versión: 2.1

Fecha: 14/01/2015

Resumen de cambios:

Cambios producidos desde la última versión

Ampliación canal PUSH

Control de difusión

Responsable:

Aprobado por:

Firma:	Fecha:

Distribución:

Referencias de archivo

Autor:

Nombre archivo:

Localización:

Contenido

1 Introducción

Este documento sirve de guía para el desarrollo de aplicaciones con envío de mesnajería SMS y PNS a terminales móviles.

Para la recepción de PNS el terminal debe tener instalada la app MEZU en su versión Android o iOS disponible en los Stores. Opcionalmente puede utilzarse un simulador en desarrollo para comprobar los envíos realizados.

El servicio se ofrece mediante un Web Service que se expone en el OSB.

2 URL's

2.1 Servicio de envío mensajería

Las URL's de los Web Services de envío SMS están publicadas en los buses de integración (Oracle Service Bus)

Desarrollo:

http://svc.intra.integracion.jakina.ejiedes.net/ctxapp/W91dSendSms

Pruebas:

http://svc.intra.integracion.jakina.ejiepru.net/ctxapp/W91dSendSms

Producción:

Intranet	http://svc.intra.integracion.jakina.ejgvdns/ctxapp/W91dSendSms
Aplicaciones desplegadas en internet.	http://svc.inter.integracion.jakina.ejgvdns/ctxapp/W91dSendSms
Aplicaciones desplegadas en extranet	http://svc.extra.integracion.jakina.ejgvdns/ctxapp/W91dSendSms
Accesos desde extranet	http://svc.integracion.ejgv.jaso/ctxweb/W91dSendSms

2.2 Url's utlidades

Las url's de utilidades y documentancíón están disponibles en la página de acceso a consolas y documentación de infraestructuras soporte y telecomunicaciones, en el enlace Latinia SMS (Desarrollo)

http://flexcloud.ejiedes.net

URL acceso plataforma de desarrollo para consulta de envíos.

http://mensajeria.movil.jakina.ejiedes.net/contracts/inicio

Empresa: INNOVUS

Usuario: DESARROLLO

Contraseña: DESARROLLO

URL simulador bandeja entrada y envío de mensajes:

http://mensajeria.movil.jakina.ejiedes.net/w91dpnsWAR/pcs.html

3 Requisitos

Para utilizar el servicio, es necesario disponer de una aplicación y un contrato de uso provisionados en la plataforma.

La salida de mensajería se realiza a través de colectores comerciales en función del operador de destino, Actualmente:

SMS: Movistar

PNS: Google (GMC) y Apple (APNS)

En desarrollo no hay colector de salida SMS con Movistar, se dispone de un operador virtual que acepta los mensajes pero no los envía al exterior.

En el caso de Google y Apple los mensajes se envían a las urls's de desarrollo de los operadores, si no se dispone de una app móvil de desarrollo no se podrán recibir los mesajes en el terminarl, aunque sí se podrán consultar en la plataforma y en la URL de simulación.

3.1 Provisión en la plataforma

Para solicitar la app en la plataforma se deberá realizar una CRQ adjuntando la plantilla de solicitud de alta

https://ataria.ejie.eus/docs/DocView.aspx?DocumentID={33a7d24a-67a1-4f3b-9571-e042ceed7f04}

En ella se indicará

- Entorno
- Código de aplicación
- Contraseña (por defecto código de aplicación)
- Descripción de la aplicación.
- Modo de uso: envío y/o recepción (por defecto sólo envío)
- Alias. Sólo para el caso de recepción

No hay plataforma en pruebas, por lo que la aplicación dada de alta en la plataforma de desarrollo valdrá también para el entorno de pruebas.

Una vez dada de alta la aplicación y el contrato en la plataforma, desde implantación se devolverá un código de contrato (ej: 2062) que junto con el código de aplicación y contraseña se utilizará para la autenticación en la plataforma.

3.2 Autenticación propia de la platarorma para uso del servicio

Los parámetros de conexión necesarios para autenticación son:

Públicos (propios de la plataforma comunes para todos):

- Empresa (Login enterprise): INNOVUS
- Usuario Latinia: innovus.superusuario
- Password Latinia: MARKSTAT

Privados (propios de cada aplicación)

- Aplicación (refProduct): <código aplicación>
- Contrato (idcontract): <código de contrato asociado a la aplicación>

- Contraseña: <por defecto código de aplicación en desarrollo>

Con esto puede formarse un xml válido para envío SMS a través de los WebServices de W91D.

Ejemplo para una aplicación Z99 con contrato 2000

```
<authenticationLatinia>
     <loginEnterprise>INNOVUS</loginEnterprise>
     <userLatinia>innovus.superusuario</userLatinia>
     <passwordLatinia>MARKSTAT</passwordLatinia>
     <refProduct>Z99</refProduct>
     <idContract>2000</idContract>
     <password>Z99</password>
</authenticationLatinia>
```

3.3 Autenticación con XLNETS

La opción más recomendable para la autenticación es el uso de XLNETS. De esta forma la aplicación se abstrae de los códigos y contraseñas de autenticación en LATINIA que serán obtenidos por el conector W91D a partir del token de sesión enviado en la petición al WebService.

Para que el conector pueda obtener los parámetros de conexión asociados al token de sesión de XLNETS, previamente se deben aprovisionar en LDAP, para ello se deberá rellenar la plantilla de provisión de elementos de seguridad XLNETS para que el SASU los de de alta.

https://ataria.ejie.eus/docs/DocView.aspx?DocumentID={33a7d24a-67a1-4f3b-9571-e042ceed7f04}

Se deberá definir un grupo con un perfil en el que se incluirá la aplicación que tiene acceso a la plataforma.

Ejemplo para aplicación Z99

PF	**Descripción en Castellano**	**Descripción en Euskera**
Z99-PF-0001	Contrato Latinia	Latinia Hitzarmena

GU	**Desc en Cast**	**Desc en Eusk**	**Perfiles (Separar ;)**
Z99-GU-0001	Envío SMS	SMS Bidalketa	Z99-PF-0001

El string de conexión se indicará en la zona de acceso a datos similar a una conexión a BBDD sólo que en este caso será de la forma <app>#idcontrato#INNOVUS#contraseña. A este string se le asociará el perfil anterior

Ejemplo para una aplicación Z99 con código contrato 2000

TO	**Código**	**BBDD**	**String de conexión**	**Subtipo**	**Perfil**
01		LATINIA	Z99#2000#INNOVUS#Z99	Z99	Z99-PF-0001

Finalmente se deberá incluir la aplicación en el grupo. Uno propio del conector para obtener los parámetros comunes y otro propio de la aplicación:

US	**Login**	**Nombre**	**Apellidos**	**Grupos a los que pertenece**
01	Z99			Z99-GU-0001

3.4 Autenticación en diferentes entornos

La plataforma LATINIA es **única para los entornos de desarrollo y pruebas**, por lo que los parámetros de aprovisionamiento inicial de la aplicación en desarrollo son válidos para pruebas, no hay que volverlos a introducir. En producción se deberá dar de alta la aplicación de nuevo.

Hay que tener en cuenta que el **código de contrato puede variar** entre desarrollo y producción en función del orden en el que se hayan ido aprovisionando las aplicaciones en ambos entornos, por lo **que es importante su revisión antes de introducirlo en XLNETS de producción.**

Por otro lado, **XLNETS sí existe en el entorno de pruebas,** por lo que aunque la plataforma LATINIA sea la misma en desarrollo y pruebas, en **XLNETS de pruebas sí que se deben aprovisionar** los datos de autenticación para que sean accesibles por las aplicaciones en pruebas.

La independencia entre los entornos de desarrollo y pruebas a efectos de uso de los conectores se realizará a través de la publicación de los WebServices del conector W91D en los distintos OSB's de desarrollo y pruebas con URL's diferentes

4 Envío de mensajes.

Hay que tener en cuenta que el envío es asíncrono. Esto quiere decir que el servicio devuelve ok o error indicando que acepta o no el mensaje para su envío, pero la aceptación no significa que este realmente llegue al destinatario.

Puede ocurrir que el terminal esté apagado, fuera de cobertura, la línea esté dada de baja etc, en cuyo caso el operador puede no informar hasta pasados unos días. Si los mensajes van sin acuse de recibo el operador no informará si definitivamente el mensaje es desechado independientemente de que el envío a la plataforma haya sido correcto.

Para más información sobre el control del estado de los envíos consultar el apartado 5.1 sobre recepción de estados.

4.1 Ejemplo básico

A continuación se muestra un ejemplo básico de envío de un mensaje SMS. Este envío puede probarse en desarrollo mediante cualquier cliente SOAP (ej SoapUI) utilizando la autenticación propia de la aplicación en la platarforma (previamente se necesita una aplicación y un contrato en la plataforma).

WSDL de desarrollo:

http://svc.extra.integracion.jakina.ejiedes.net/ctxapp/W91dSendSms?WSDL

En negrita aparece el XML con el menaje.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:w91d="http://w91d">
<soapenv:Header>
<authenticationLatinia>
      <loginEnterprise>INNOVUS</loginEnterprise>
      <userLatinia>innovus.superusuario</userLatinia>
      <passwordLatinia>MARKSTAT</passwordLatinia>
      <refProduct>A00</refProduct>
      <idContract>2000</idContract>
      <password>xxxxxxx</password>
      </authenticationLatinia>
</soapenv:Header>
<soapenv:Body>
<StringInput xmlns="http://w91d">
<![CDATA[<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<PETICION>
<LATINIA>
<MENSAJES>
<MENSAJE_INFO ACUSE="S">
      <TEXTO>SMS HOLA MUNDO</TEXTO>
      <GSM_DEST>688000000</GSM_DEST>
</MENSAJE_INFO>
</MENSAJES>
</LATINIA>
</PETICION>]]>
</StringInput>
</soapenv:Body>
</soapenv:Envelope>
```

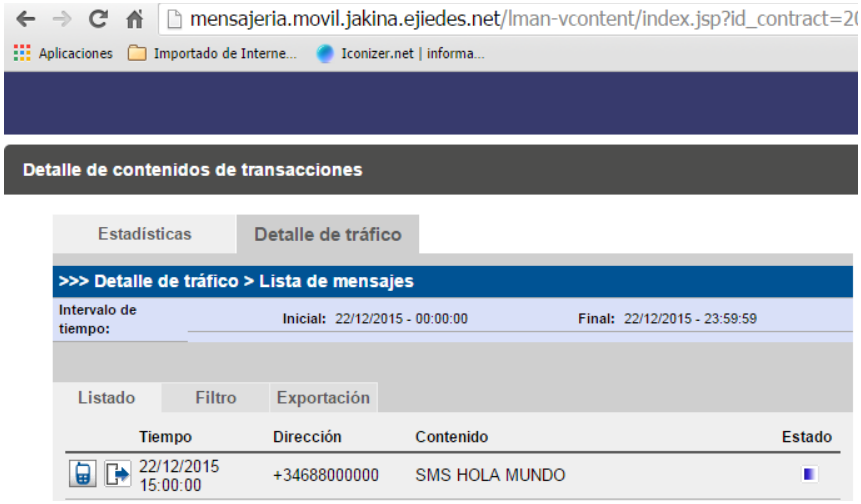
En el caso de utilizar **XLNETS,** en la cabecera del mensaje aparecería el token de sesión de la aplicación correspondiente.

La respuesta sería similar a:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header/>
<S:Body xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<StringOutput xmlns="http://w91d"><![CDATA[
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<PETICION><MENSAJE NUM="1">
<TELEFONO NUM="688000000">
<RESULTADO>OK</RESULTADO>
<IDENTIFICADOR>dGSjY3W0RHqDomGbLoxpnhRY</IDENTIFICADOR>
</TELEFONO>
</MENSAJE>
</PETICION>]]>
</StringOutput>
</S:Body>
</soapenv:Envelope>
```

El resultado puede comprobarse en la consola de desarrollo (ver Url's utiliadades en 2.2)

Análisis->Detalle del contenido de las transacciones:



4.2 Formato completo del mensaje de salida

A continuación se detalla el formato del xml del mensaje de salida para SMS/PNS

```
<?xml version="1.0" encoding="UTF-8"?>
<PETICION>
	<LATINIA>
		<MENSAJES>
			<MENSAJE_INFO ACUSE="S">
				<TS_DIFERIDO>620000000</TS_DIFERIDO> <!-- TimeStamp (long)-->
				<TEXTO_SMS>SMS prueba mensaje viernes 2</TEXTO_SMS>
				<TEXTO_PNS>SMS prueba mensaje viernes 2</TEXTO_PNS>
				<PRIVADO>Mensaje privado a consultar mediante la app</PRIVADO>
				<GSM_DEST>659000001,666000001</GSM_DEST>
			</MENSAJE_INFO>
			<MENSAJE_INFO ACUSE="N">
				<TS_EXPIRE>10</TS_EXPIRE> <!-- Minutos de vida del mensaje-->
				<TEXTO>Hola mundo!!!</TEXTO>
				<GSM_DEST>659000001</GSM_DEST>
			</MENSAJE_INFO>
```

```
            </MENSAJES>
        </LATINIA>
</PETICION>
```

A continuación se explican las distintas etiquetas, los formatos de los datos que deben enviarse:

- **MENSAJES**: Lista de MENSAJE_INFO (mensajes) que se desean enviar.
- **MENSAJE_INFO**: información del mensaje que se quiere enviar.
  - **ACUSE**: S o N. S → Se pide notificación de entrega al proveedor. N → No se pide notificación de entrega al proveedor.
- **TS_DIFERIDO**: Timestamp en milisegundos (long) con la fecha concreta en la que se quiere entregar el mensaje.
- **TS_EXPIRE**: Minutos de vida del mensaje (int). Fecha a partir de la cual el mensaje ya no se va a enviar.
- **TEXTO**: Sección con el texto del mensaje. Se conserva por compatibilidad con formato anterior. Se recomienda utilizar TEXTO_SMS
- **TEXTO_SMS**: Texto específico para SMS es equivalente a TEXTO
- **TEXTO_PNS**: Texto específico notificación PUSH a motrar en el terminal, por defecto se copia el contenido de TEXTO o TEXTO_SMS.
- **PRIVADO**: Parte privada del mensaje. Sólo se puede consultar mediante la app. Por defecto se copia el contenido de TEXTO_PNS.
- **GSM_DEST**: Lista de los números de teléfono destinatarios. Como máximo se pueden enviar 150 y tienen que ir separados por comas.

4.3 Formato respuesta

El método **sendSms** devuelve un xml con el siguiente formato:

```
<?xml version="1.0" encoding="UTF-8"?>
<PETICION>
        <MENSAJE NUM="1">
                <TELEFONO NUM="659000001">
                        <RESULTADO>OK</RESULTADO>
                        <IDENTIFICADOR>UGsiZ7E1naZX/Uey32A1hFUq</IDENTIFICADOR>
                </TELEFONO>
                <TELEFONO NUM="666000001">
                        <RESULTADO>OK</RESULTADO>
                        <IDENTIFICADOR>UGsiZ7E2efSshUey32A1mU7o</IDENTIFICADOR>
                </TELEFONO>
        </MENSAJE>
        <MENSAJE NUM="2">
                <TELEFONO NUM="659000001">
                        <RESULTADO>ERROR</RESULTADO>
                        <CODIGO_ERROR>301</CODIGO_ERROR>
                        <MENSAJE_ERROR>El mensaje ha expirado</MENSAJE_ERROR>
                </TELEFONO>
        </MENSAJE>
</PETICION>
```

Este xml de respuesta nos indica cuales son los mensajes que han llegado de manera correcta a la plataforma y cuáles no. De este modo la aplicación usuaria de este servicio tendrá la información correspondiente al estado de los mensajes y si la operación ha terminado de manera correcta.

A continuación se explica el formato del xml devuelto por el método sendSms del servicio web:

- **MENSAJE**: Si un mensaje es enviado a varios números de teléfono nos devuelve una lista con el resultado de cada uno de ellos.

- **NUM**: número del mensaje que se ha enviado. Es el número de orden en el envío.
- **TELEFONO**: Destinatario
- **NUM**: Número de teléfono al que se ha enviado el mensaje.
- **RESULTADO**: OK o ERROR. Si el mensaje se ha entregado bien se devuelve un OK. Si ha sucedido algún error se devuelve ERROR.
- **IDENTIFICADOR**: Si el resultado ha sido OK, se devuelve el identificador único del mensaje, id interno del mensaje en la plataforma (para la Q68) y en identificador externo, no único, en el caso de las aplicaciones no Q68. En este último caso Identificador + Num de teléfono será la manera de identificar cada sms enviado.
- **CODIGO_ERROR**: Código asignado al error que se ha producido.
- **MENSAJE_ERROR**: Descripción del error que se ha generado.

4.3.1. Error en formato mensaje salida

Puede ocurrir que el mensaje de respuesta SOAP no contenga elementos MENSAJE, en cuyo caso indicará un error en el formato de mensaje de salida;

Ej:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <S:Body xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <StringOutput xmlns="http://w91d">&lt;?xml version="1.0"
encoding="UTF-8" standalone="no"?>&lt;PETICION/></StringOutput>
   </S:Body>
</soapenv:Envelope>
```

4.3.2. Errores de sistema

Si ocurre algún error de sistema, este apecerá en la descripción del error. A continuación aparece un error de autetnicación

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header/>
   <S:Body xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <StringOutput xmlns="http://w91d"><![CDATA[<?xml version="1.0"
encoding="UTF-8" standalone="no"?><PETICION><MENSAJE NUM="1"><TELEFONO
NUM="688000000"><RESULTADO>ERROR</RESULTADO><CODIGO_ERROR>500</CODIGO_ERROR>
<MENSAJE_ERROR>Error desconocido.. Message: null. Cause:
javax.security.auth.login.FailedLoginException:
[Security:090304]Authentication Failed: User W91DPNS
javax.security.auth.login.FailedLoginException:
[Security:090302]Authentication Failed: User W91DPNS
denied</MENSAJE_ERROR></TELEFONO></MENSAJE></PETICION>]]></StringOutput>
   </S:Body>
</soapenv:Envelope>
```

5 Recepción de estados y mensajes.

La recepción de estados y mensajes se realiza a través del PROPAGADOR/ENRUTADOR utilizado en EJIE para distribuir mensajería JMS. Es un sistema publicación/suscripción en el que las aplicaciones se suscriben a mensajes que otros publican.

En escte caso la publicación de los mensajes la realiza el conector W91D desplegado en la plataforma de mensajería y las aplicaciones que lo deseen se suscribirán a los mensajes que para su tratamiento.

Hay tres tipos de eventos:

- **LATINIA_ESTADO:** se informa de los códigos de retorno de la plataforma por cada envío que se realiza. Estos códigos dan la trazabilidad del estado de los envíos.
- **LATINIA_MENSAJE**: se trata de mensajes que el usuario final envía a nuestra plataforma.
- **LATINIA_REGISTRO**: se propaga cuando se de detecta un registro correcto de una app en la plataforma.

El conector W91D se encarga de recoger los eventos generados en la plataforma de mensajería y propagarlos a través del propagador/enrutador a las aplicaciones suscritas. En este escenario, cada aplicación se conecta a una cola propia donde se le enviarán los estados y mensajes a los que esté suscrita. (Ver documentación Platea Integración Eventos en http://flexcloud.ejiedes.net)

5.1 Recepción de estados. LATINIA_ESTADO

Como norma general, si se van a tratar eventos de recepción conviene poder identificarlos con un envío realizado para de esta manera poder casar los envíos con los eventos de recepción. Hay que tener en cuenta que el sistema es asíncrono, por lo que puede que en envíos masivos se empiecen a recibir eventos de recepción antes de que el envío se haya terminado completanemte.

En este sentido la recomendación es:

- Por cada envío, almacenar el id y el destinatario del mensaje.
- Almacenar los eventos de recepción forma asíncrona independientemente de que se tenga registrado el envío.

A continuación se muestra un ejemplo de un xml de estado publicado en el propagador.

El contenido de la etiqueta **<xmlValue>** es el evento que indica el estado del mensaje SMS enviado. En este caso se ha enviado con notificación de entrega y el estado ENVIADO=1 Y ESTADO=4 indica, ha sido entregado a su destinatario (ver códigos de estado en apartado Errorea: ez da erreferentziaren iturburua aurkitu)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<NotificationBody xmlns="com/ejie/notification/xml">
	<Property>
		<Name>EventWhat</Name>
		<Value>LATINIA_ESTADO</Value>
	</Property>
	<Property>
		<Name>EventWho</Name>
		<Value>SMS</Value>
	</Property>
	<Property>
		<Name>EventTipology</Name>
		<Value>LATINIA</Value>
	</Property>
```

```
        <Property>
                <Name>EventCorrelationId</Name>
                <Value>974846.1452594262220.0</Value>
        </Property>
        <Property>
                <Name>EventEntity</Name>
                <Value>EJIE</Value>
        </Property>
        <Property>
                <Name>EventTimeStamp</Name>
                <Value>1452594261867</Value>
        </Property>
        <Property>
                <Name>REFPRODUCT</Name>
                <Value>W91DTEST</Value>
        </Property>
        <xmlValue>
                <?xml version="1.0" encoding="UTF-8"?>
                        <EVENT>
                        <IDENTIFICADOR>UGtrTbSQm75wjEey32A218Zg</IDENTIFICADOR>
                                <TELEFONO>666000001</TELEFONO>
                                <ESTADOS>
                                        <ENVIADO>1</ENVIADO>
                                        <ESTADO>4</ESTADO>
                                        <DESCRIPCION>Proveedor notifica que ha sido
entregado el mensaje.</DESCRIPCION>
                                </ESTADOS>
                                <TS_EVENTO>020311175501</TS_EVENTO>
                        </EVENT>
        </xmlValue>
</NotificationBody>
```

Las etiquetas son:

- **IDENTIFICADOR**: Identificador del mensaje.
- **TELEFONO**: Número del teléfono móvil destinatario del mensaje.
- **ENVIADO:** Situación general del mensaje, un valor negativo indicará un error (ver 5.1.1)
- **ESTADO**: Código del estado en el que esta el mensaje (ver 5.1.1).
- **DESCRIPCION**: Descripción del estado.
- **TS_EVENTO**: Timestamp del del evento. (Formato: DDMMYYHHMMSS).

5.1.1. *Códigos* de estado

Los mensajes movidos por la plataforma pasan en su ciclo de vida por diferentes estados. Para cada uno de los estados se genera un evento que identifica el estado del mensaje y es publicado en el propagador/enrutador

Cada evento de estado se dientifica por dos campos: El primero campo (ENVIADO) identifica la situación general del mensaje, **un valor negativo indicará un error**. El segundo campo (ESTADO) identifica el estado concreto en el que esta el mensaje

En la consola de administración, estos dos valores se representan como un par de números separados por una coma o una barra:(1,2), (-3/1), etc…

- **ENVIADO = 1**

Enviado al exterior de la Plataforma

ENVIADO	ESTADO	DESCRIPCIÓN
1	2	Entregado al Proveedor
1	4	Entrega confirmada a usuario

- **ENVIADO = -2**

ERROR: Rechazado por el proveedor

ENVIADO	ESTADO	DESCRIPCIÓN
-2	1	Rechazado por el Proveedor antes de la entrega
-2	2	Expirado
-2	3	Rechazado por el Proveedor. Mensaje incorrecto o inapropiado
-2	4	Destinatario incorrecto. El Proveedor no acepta el destinatario.
-2	5	Cancelado por el Proveedor (sólo e-mail)

- **ENVIADO = -3**

ERROR: Rechazado por la plataforma

ENVIADO	ESTADO	DESCRIPCIÓN
-3	1	Envío Fallido a La LIMSP
-3	2	Sin regla de negocio (no figuran en las estadísticas)
-3	3	Parámetros incorrectos (no figuran en las estadísticas)
-3	4	Trama Incorrecta (no figuran en las estadísticas)
-3	5	Crédito insuficiente
-3	6	Remitente no autorizado
-3	7	Destinatario no autorizado
-3	8	Error de conexión
-3	9	Expirado por Plataforma

5.1.2. Identificación del canal de salida

En el evento aparece en el campo <TELEFONO> el destinatario y canal por el que la plataforma ha enviado el mensaje. En el caso en el que mensaje haya salido por el canal SMS el valor se corresponderá con el número de teléfono del destinatario ej: <TELEFONO>688888888</TELEFONO>

En el caso de que el canal de salida haya sido PUSH, el valor del campo <TELEFONO> indicará el token corresponeiente al detinatario y el proveedor separados por '@'

Los formatos son:

CANAL	Valor
SMS	<TELEFONO>688888888</TELEFONO>
PUS Apple	<TELEFONO>…6922d928c8ea0eb8e4@**apple**</TELEFONO>
PUS Google	<TELEFONO…eKoWiKAtmv1TJu-N34@**google**</TELEFONO>

5.2 Recepción de mensajes. LATINIA_MENSAJE

A continuación se muestra un ejemplo de un evento generado al recibir un mensaje de un usuario. Este mensaje puede ser SMS o una respuesta a una PNS

El contenido de la etiqueta **<xmlValue>** es el contenido del mensaje. Todos los mensajes tienen una propiedad dinámica **ALIAS** que puede utilizarse para realizar las suscripciones

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<NotificationBody xmlns="com/ejie/notification/xml">
	<Property>
		<Name>EventWhat</Name>
		<Value>LATINIA_MENSAJE</Value>
	</Property>
	<Property>
		<Name>EventWho</Name>
		<Value>SMS</Value>
	</Property>
	<Property>
		<Name>EventTipology</Name>
		<Value>LATINIA</Value>
	</Property>
	<Property>
		<Name>EventCorrelationId</Name>
		<Value>974846.1452594262011.0</Value>
	</Property>
	<Property>
		<Name>EventEntity</Name>
		<Value>EJIE</Value>
	</Property>
	<Property>
		<Name>EventTimeStamp</Name>
		<Value>1452594261867</Value>
	</Property>
	<Property>
		<Name>ALIAS</Name>
		<Value>W91DTEST</Value>
	</Property>
	<xmlValue>
		<?xml version="1.0" encoding="UTF-8"?>
			<MENSAJE>
				<IDENTIFICADOR>UGtrTbSQm75wjEey32A218Zg</IDENTIFICADOR>
				<TELEFONO>666000001</TELEFONO>
				<TS_MENSAJE>020311175501</TS_MENSAJE>
				<TEXT_MENSAJE>educa 001</TEXT_MENSAJE>
			</MENSAJE>
	</xmlValue>
</NotificationBody>
```

Las etiquetas del mensaje son:

- **IDENTIFICADOR**: Identificador del mensaje.
- **TELEFONO**: Número GSM del teléfono móvil remitente.
- **TEXT_MENSAJE**: Texto del mensaje del usuario móvil
- **TS_MENSAJE**: Timestamp de la recepción del mensaje en la plataforma Latinia. (Formato: DDMMYYHHMMSS).

5.3 Recepción de evento de registro. LATINIA_REGISTRO

Cuando se detecta que un usuario ha completado correctamente la instalación y registro de una app en la plataforma, se generá un evento indicando el teléfono, el proveedor y el timestamp.

El mensaje contendrá una propiedad dinámica **APPREF** que puede utilizarse para realizar las suscripciones y que indicará la referencia interna de la app móvil.

El contenido de la etiqueta **`<xmlValue>`** es el contenido del mensaje.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<NotificationBody xmlns="com/ejie/notification/xml">
```

```
        <Property>
                <Name>EventWhat</Name>
                <Value>LATINIA_REGISTRO</Value>
        </Property>
        <Property>
                <Name>EventWho</Name>
                <Value>SMS</Value>
        </Property>
        <Property>
                <Name>EventTipology</Name>
                <Value>LATINIA</Value>
        </Property>
        <Property>
                <Name>EventCorrelationId</Name>
                <Value>974846.1452594262220.0</Value>
        </Property>
        <Property>
                <Name>EventEntity</Name>
                <Value>EJIE</Value>
        </Property>
        <Property>
                <Name>EventTimeStamp</Name>
                <Value>1452594261867</Value>
        </Property>
        <Property>
                <Name> APPREF</Name>
                <Value>gvi</Value>
        </Property>
        <xmlValue>
                <?xml version="1.0" encoding="UTF-8"?>
                        <MENSAJE>
                                <TELEFONO>+34666000001</TELEFONO>
                                <PROVEEDOR>Google</PROVEEDOR>
                                <TS_EVENTO>020311175501</TS_EVENTO>
                        </MENSAJE>
        </xmlValue>
</NotificationBody>
```

Las etiquetas del xmlValue son:

- **TELEFONO**: Número del teléfono móvil registrado (llevará código de país por delante).
- **PROVEEDOR:** Proveedor del Sistema Operativo del terminal: Google o Apple
- **TS_EVENTO**: Timestamp del del evento. (Formato: DDMMYYHHMMSS).

6 Envío batch desde back-end

Para el envío masivo de mensajes se recomienda el uso de procesos en back-end a ejecutar con control-m ó k31.

En la plataforma existen tres prioridades a la hora de enviar los mensajes al proveedor:

Prioridad ALTA: utilizada para envíos urgentes, alarmas, avisos de seguridad, etc

Prioridad NORMAL: es la prioridad utilizada por defecto en las aplicaciones

Prioridad BAJA: es la prioridad a utilizar en el caso de envíos masivos.

Para el caso de los envíos batch masivos es recomendable disponer de un contrato con prioridad **BAJA** para interferir lo menos posible con la salida de mensajes on-line del resto de aplicaciones.

A continuación se muestran dos ejemplos con curl y con telnet para envío de mensajes desde Shell ksh en máquinas de backend.

6.1 Envío con curl

En este caso se reciben como parámetros los teléfonos separados por comas y el mensaje. Se monta directamente el mensaje SOAP y se realiza un post mediante curl

```
#!/bin/ksh
TELEFONOS=$1
MENSAJE=$2

if [ -z $TELEFONOS ]; then
        echo "Falta parámetro con teléfono(s)"
        exit 1
fi

if [ -z $MENSAJE ]; then
        echo "Falta parámetro con mensaje"
        exit 2
fi

##################################
#Webservice w91d para envio SMS
#Desarrollo
W91DSENDSMS=http://svc.intra.integracion.jakina.ejiedes.net/ctxapp/W91dSendSms

#Autenticacion
APLICACION=W91DTEST
IDCONTRACT=2022
PASSWORD=xxxxxxxx

#XML SOAP A enviar
DATOS="<soapenv:Envelope xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:w91d=\"http://w91d\">"
DATOS=$
{DATOS}"<soapenv:Header><authenticationLatinia><userLatinia>innovus.superusuario</us
erLatinia><passwordLatinia>MARKSTAT</passwordLatinia><refProduct>$
{APLICACION}</refProduct><idContract>$
{IDCONTRACT}</idContract><loginEnterprise>INNOVUS</loginEnterprise><password>$
{PASSWORD}</password></authenticationLatinia></soapenv:Header>"
DATOS=${DATOS}"<soapenv:Body><StringInput xmlns=\"http://w91d\">"
DATOS=${DATOS}"<![CDATA[<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
DATOS=${DATOS}"<PETICION><LATINIA><MENSAJES><MENSAJE_INFO ACUSE=\"S\">"
DATOS=${DATOS}"<TEXTO>${MENSAJE}</TEXTO>"
DATOS=${DATOS}"<GSM_DEST>${TELEFONOS}</GSM_DEST>"
DATOS=${DATOS}"</MENSAJE_INFO></MENSAJES></LATINIA></PETICION>]]>"
```

```
DATOS=${DATOS}"</StringInput></soapenv:Body>"
DATOS=${DATOS}"</soapenv:Envelope>"


echo ""
echo "Enviando..."
echo "URL: ${W91DSENDSMS}"
echo "SOAP:  ${DATOS}"
echo ""

curl -X POST --data "${DATOS}" "${W91DSENDSMS}"

SALIDA=$?

echo ""
echo ""
echo "Enviado. Retorno: $SALIDA"
echo ""

exit  $SALIDA
```

6.2 Envío con telnet

Ídem anterior pero realizando la llamada con telnet. Puede utilizarse en servidores en los que no sea posible utilizar curl.

```
#!/bin/ksh
TELEFONOS=$1
MENSAJE=$2

if [ -z $TELEFONOS ]; then
        echo "Falta parámetro con teléfono(s)"
        exit 1
fi

if [ -z $MENSAJE ]; then
        echo "Falta parámetro con mensaje"
        exit 2
fi

#################################
#Webservice w91d para envio SMS
ENDPOINT=/ctxapp/W91dSendSms
#Host Desarrollo
SERVIDOR="svc.intra.integracion.jakina.ejiedes.net"
PUERTO=80

echo "SERVIDOR: $SERVIDOR"
echo "PUERTO: $PUERTO"

#Autenticacion en la plataforma
APLICACION=W91DTEST
IDCONTRACT=2022
PASSWORD=xxxxxxxx

#XML SOAP A enviar
DATOS="<soapenv:Envelope xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:w91d=\"http://w91d\">"
DATOS=$
{DATOS}"<soapenv:Header><authenticationLatinia><userLatinia>innovus.superusuario</us
erLatinia><passwordLatinia>MARKSTAT</pass
wordLatinia><refProduct>${APLICACION}</refProduct><idContract>$
{IDCONTRACT}</idContract><loginEnterprise>INNOVUS</loginEnterprise><p
assword>${PASSWORD}</password></authenticationLatinia></soapenv:Header>"
```

```
DATOS=${DATOS}"<soapenv:Body><StringInput xmlns=\"http://w91d\"><![CDATA[<?xml
version=\"1.0\" encoding=\"UTF-8\"?><PETICION><LATINI
A><MENSAJES><MENSAJE_INFO><TEXTO><![CDATA[${MENSAJE}]]]]><!
[CDATA[</TEXTO><GSM_DEST>${TELEFONOS}</GSM_DEST></MENSAJE_INFO></MENSAJE
S></LATINIA></PETICION>]]></StringInput></soapenv:Body>"
DATOS=${DATOS}"</soapenv:Envelope>"

echo $DATOS

#CABECERAS http
CABECERAS="POST ${ENDPOINT} HTTP/1.1\n"
CABECERAS=${CABECERAS}"Content-Type: text/xml;charset=UTF-8\n"
CABECERAS=${CABECERAS}"SOAPAction: \"\"\n"
CABECERAS=${CABECERAS}"User-Agent: Jakarta Commons-HttpClient/3.1\n"
CABECERAS=${CABECERAS}"Host: ${SERVIDOR}\n"
CABECERAS=${CABECERAS}"Content-Length: ${#DATOS}\n"

#Se vuelca la salida a un log y se saca por pantalla
FECHA=`date +"%Y_%m_%d_%H_%M_%S"`
NOMFICLOG="SMS$$_"${FECHA}.log
DIRLOG=/tmp/
FICLOG=${DIRLOG}${NOMFICLOG}

(
sleep 1
echo ${CABECERAS}
echo ${DATOS}
sleep 5
) | telnet ${SERVIDOR} ${PUERTO} >$FICLOG

RESULT=`grep "<RESULTADO>OK</RESULTADO>" $FICLOG`
echo $RESULT
cat $FICLOG
rm $FICLOG

if [ "$RESULT" == "" ]; then
        exit 1
fi
exit 0
```

7 Envío dual Mensajería SMS/PNS

Se dispone de una app MEZU que el usuario puede descargarse en el terminal para recibir notificaciones con posibilidad de respuesta.

A priori las aplicaciones que envían mensajes desconocen si el destinatario tiene o no la app en su terminal, por lo que pueden enviar mensajes contemplando esta posibilidad.

El mensaje que se envía a los terminales puede cntener tres partes:

1. **Pate pública SMS**: Se envía de forma habitual al operador de telefonía GSM. Es universal para todos los teléfonos con una línea válida
2. **Parte pública PNS**: Esta parte se envía directamente al proveedor (Google, Apple) y es éste el que se encarga de localizar el terminal y descargarle la notificación. Es similar al SMS y se muestra en el terminal cuando se recepciona.
3. **Parte privada**: Esta parte no se envía al proveedor, y sólo sale de los servidores en el caso de que el usuario lo solicite a través de la aplicación en el terminal. El envío es directo entre nuestros servidores y el terminal sin pasar por el proveedor.

En la provisión de los contratos se establecerá por defecto el envío de PNS si el usuario tiene instalada la app, no obstante pueden modificarse las condiciones de envío para:

- **Enviar sólo SMS**: No se quiere que los usuarios utilicen la aplicación o no es posible.
- **Enviar sólo PNS**: Envíos masivos en los que se quiere evitar costes envío SMS
- **Enviar ambos**: Alarmas, avisos de seguridad etc.

A continuación aparece un ejemplo del XML con un mensaje que se envía al Web Service del conector W91D en el que se envían las tres partes.

```
<PETICION>
      <LATINIA>
      <MENSAJES>
            <MENSAJE_INFO ACUSE="S">
            <GSM_DEST>644318777</GSM_DEST>
            <TEXTO_SMS>SMS Hola mundo</TEXTO_SMS>
            <TEXTO_PNS>PNS Hola mundo</TEXTO_PNS>
            <PRIVADO>Hola mundo. Esto es la parte privada</PRIVADO>
            </MENSAJE_INFO>
      </MENSAJES>
      </LATINIA>
</PETICION>
```

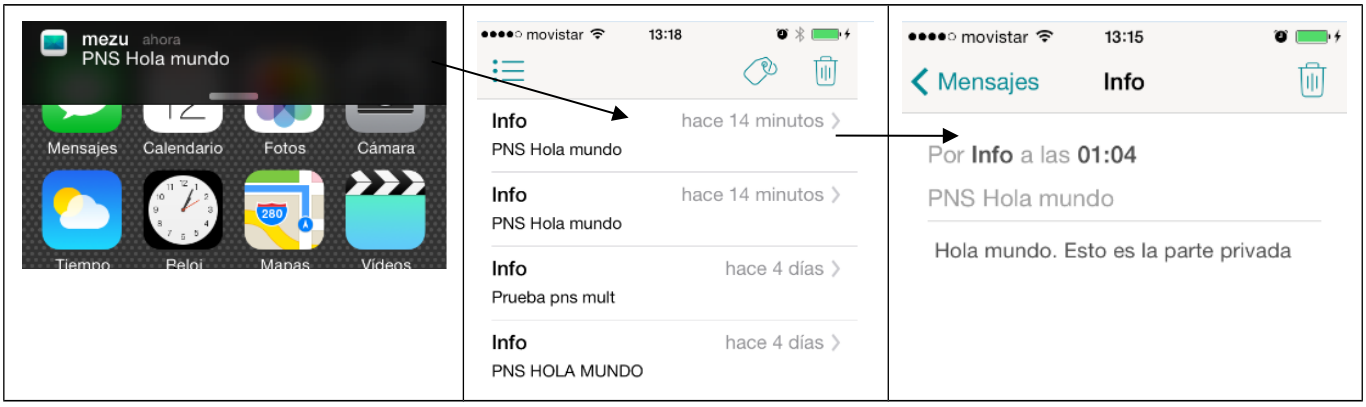
En el caso de que el usuario no disponga de la aplicación en el terminal, se le enviará un SMS tradicional con la parte pública TEXTO_SMS, por lo que hay que tener en cuenta las limitaciones de este formato (160 caracteres).

En el caso de que asociado al número GSM se detecte que el terminal tenga instalada la aplicación, se enviará una notificación a través del proveedor correspondiente. En este caso el terminal mostrará el mensaje PNS y activará la APP para visualizar el contenido privado.

El contenido privado es opcional, y en el caso de no existir, por defecto se copiará el contenido público TEXTO_SMS ó TEXTO_PNS

Etiqueta	Obligatorio	Contenido
TEXTO_SMS	Sí	El indicado en el xml
TEXTO_PNS	No	Si no existe se copia el contenido de TEXTO_SMS
PRIVADO	No	Si no existe, se copia el contenido de TEXTO_PNS y si no existe TEXTO_PNS, se copia el de TEXTO_SMS

Ejemplo de un terminal con la aplicción MEZU que recibe la notificación PNS:

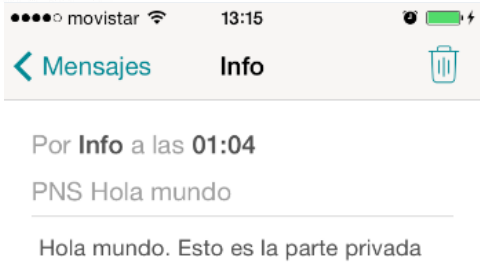


Ejemplo del mismo mensaje pero a un terminal que no tiene la app MEZU. En este caso se recibe un SMS.

7.1 Formato mensaje Privado

La finalidad del texto privado es ampliar el contenido de la notificación sin las restricciones de los SMS, se puede incluir información sensible evitando el paso por servidores de terceros, así como ofrecer la posibilidad de envío de respuestas

En el caso de que el mensaje incluya contenido privado, este puede consistir en un texto simple que se mostrará sin más en el terminal del usuario:



Opcionalmente, el texto privado puede ser un JSON que habilitará a la aplicación para mostrar al usuario diferentes opciones para enviar respuestas al mensaje.

A continuación se describe el formato del mensaje:

7.2 JSON mensaje privado. Opción de respuesta

En el caso de que el mensaje privado sea un JSON, seguirá el esquema que se describe a continuación. En caso contrario se supondrá un texto libre que no incluirá opciones de respuestas.

Opcionalmente puede haber mensajes con el JSON correcto pero que no incluyan respuestas, por lo que a efectos prácticos de cara al usuario serán como mensajes de texto.

Campos JSON:

Parámetro	**Descripción**	
private	Etiqueta obligatoria del JSON correspondiente al texto privado.	
	categoria	Etiqueta opcional que indica la categoría del mensaje para organizar la vista en la app. El menaje colgará dentro de esta categoría.
		eu: Texto de la categoría en euskera (1)
		es: Texto de la categoría en castellano (1)
	texto	Etiqueta obligatoria para el texto privado del mensaje a mostrar al usuario en función del idioma (1) Debe aparecer por lo menos en español (es).
		eu: Texto privado en el idioma euskera (1)
		es: Texto privado en castellano (1)
	respuesta	Parámetros para construir la respuesta al mensaje. Opcional. Si no existe se mostrará el mensaje sin opciones de respuesta.
		alias: Alias del mensaje de respuesta. Se corresponde con la palabra clave en el caso de los SMS's

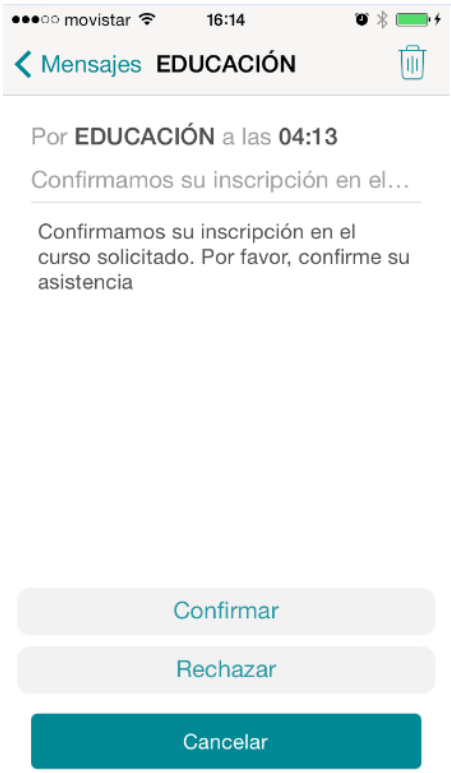
		valores	Posibles respuestas al mensaje	
			codigo	Código de la respuesta que se deberá enviar asociado al alias de la respuesta. Se corresponde con el texto del SMS en el caso de envío con palabra clave del tipo : “ALIAS CODIGO”
			eu	Opción a mostrar en euskera (1)
			es	Opción a mostrar en castellano (1)

(1) Inicialmente la app estará en euskera/castellano, pero en los mensajes podría extenderse el número de idiomas, (en, fr etc),

Ejemplo:

```
{
  "private": {
   "categoria": {
      "eu": "HEZKUNTZA",
      "es": "EDUCACIÓN"
    },
    "texto": {
      "eu": "Eskatutako kurtsoan izen-emate baieztatzen dizuegu. Zure
etortzea baieztatu, mesedez",
      "es": "Confirmamos su inscripción en el curso solicitado. Por
favor, confirme su asistencia"
    },
    "respuesta": {
      "alias": "AA4455CONF",
      "valores": [
        {
          "codigo": "52232323P 1",
          "eu": "Baietztatu",
          "es": "Confirmar"
        },
        {
          "codigo": "52232323P 2",
          "eu": "Ezeztatu",
          "es": "Rechazar"
        }
      ]
    }
  }
}
```

Con esto la aplicación mostrará el texto y las opciones en función de los valores del JSON y del idioma de la aplicación:



Con cualquiera de las dos respuestas (Confirmar o Rechazar) la app MEZU enviará un mensaje de respuesta que se propagará a través del PROPAGADOR/ENRUTADOR a las aplicaciones suscritas (ver apartado 5.2).

En este ejemplo se recibirá un mensaje con **ALIAS=AA4455CONF** y el texto del mensaje correspondiente a la respuesta seleccionada por el usuario, en este caso **1 ó 2**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<NotificationBody xmlns="com/ejie/notification/xml">
	<Property>
		<Name>ALIAS</Name>
		<Value>AA4455CONF</Value>
	</Property>
	<xmlValue>
		<?xml version="1.0" encoding="UTF-8"?>
			<MENSAJE>
				<IDENTIFICADOR>UGtrTbSQm75wjEey32A218Zg</IDENTIFICADOR>
				<TELEFONO>666000001</TELEFONO>
				<TS_MENSAJE>020311175501</TS_MENSAJE>
				<TEXT_MENSAJE>52232323P 1</TEXT_MENSAJE>
			</MENSAJE>
	</xmlValue>
</NotificationBody>
```

8 ANEXO 1. Ejemplo de cliente JAX-WS con eclipse

Realizaremos un cliente ws en eclipse a partir del WSDL de desarrollo e introduciremos un handler para enviar la autenticación en la cabecera del mensaje SOAP.

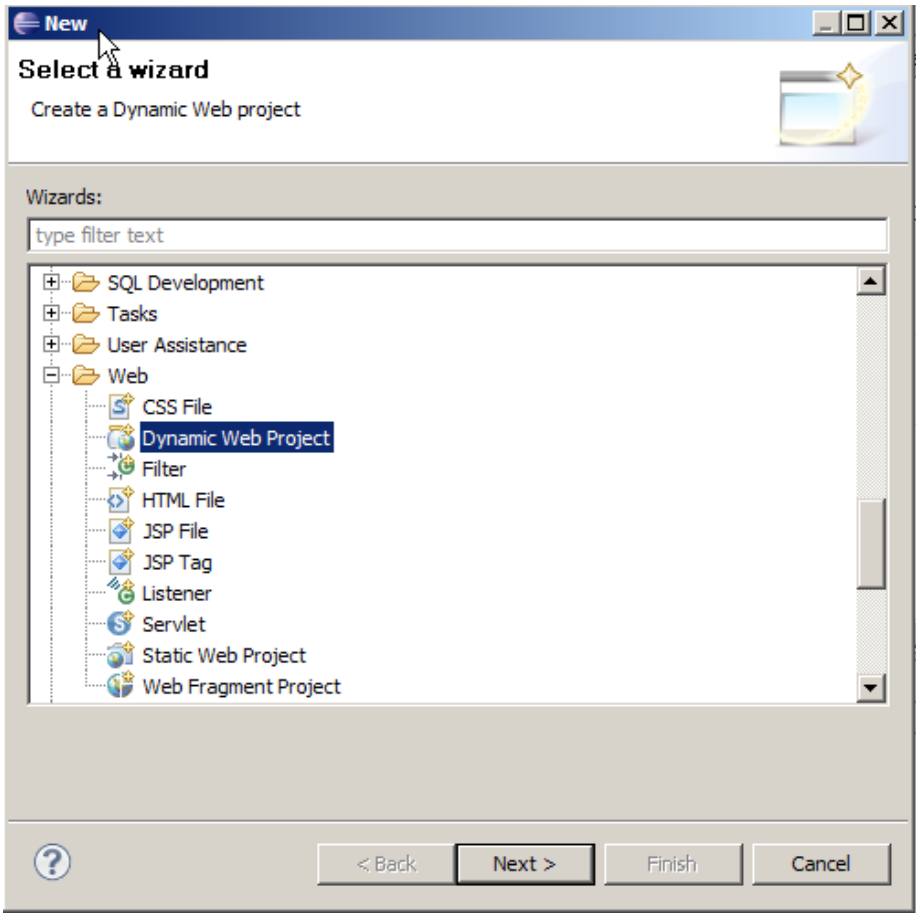
8.1 Requisitos

Previamente es necesario tener instalado el runtime Apache CXF:

http://help.eclipse.org/mars/index.jsp?topic=%2Forg.eclipse.jst.ws.cxf.doc.user%2Ftasks%2Fant_tasks.html

8.2 Creación del cliente

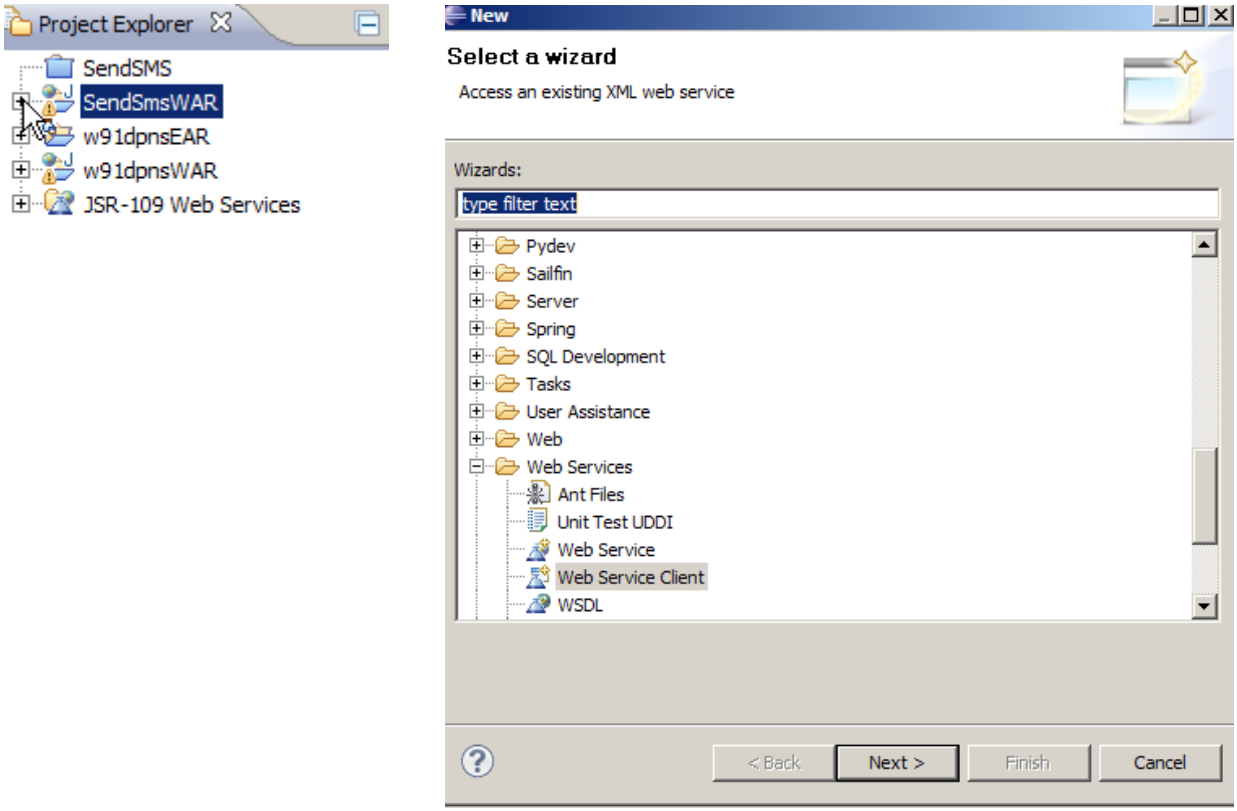
Crear un nuevo proyecto **"Dinamic Web Project"**



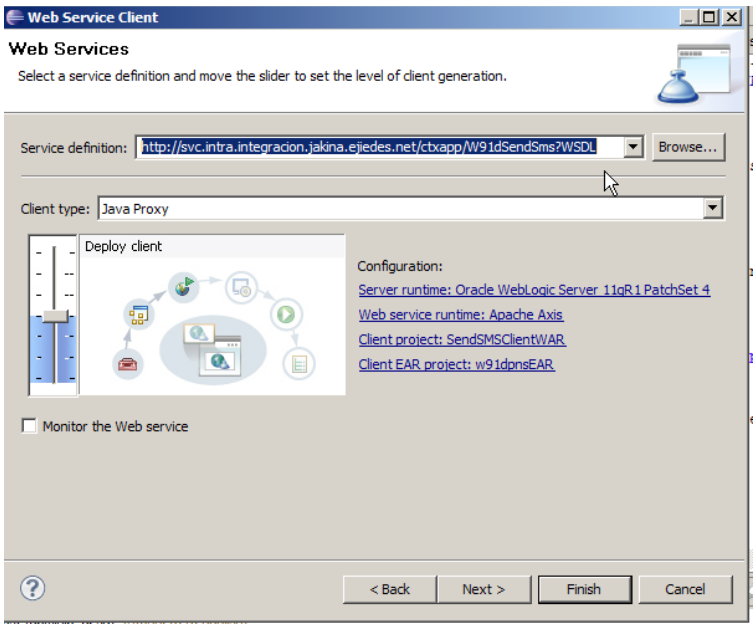
En el ejemplo crearemos un proyecto **SendSmsWAR**

Una vez creado Seleccionar el proyecto y agregaremos un cliente webservice

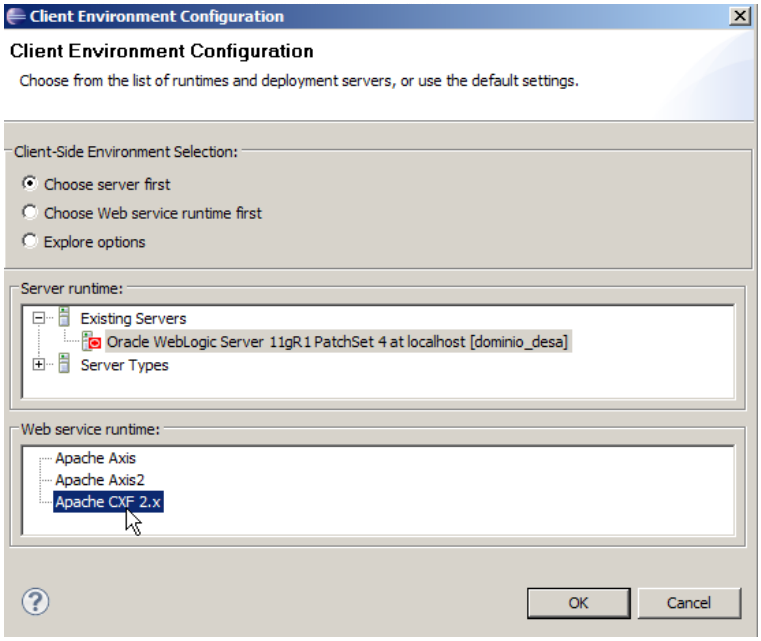
Botón derecho->New->Web Services->Web Service Client



En la casilla Service Definition introducir la URL del WSDL del servicio del OSB de desarrollo:
http://svc.extra.integracion.jakina.ejiedes.net/ctxapp/W91dSendSms?WSDL

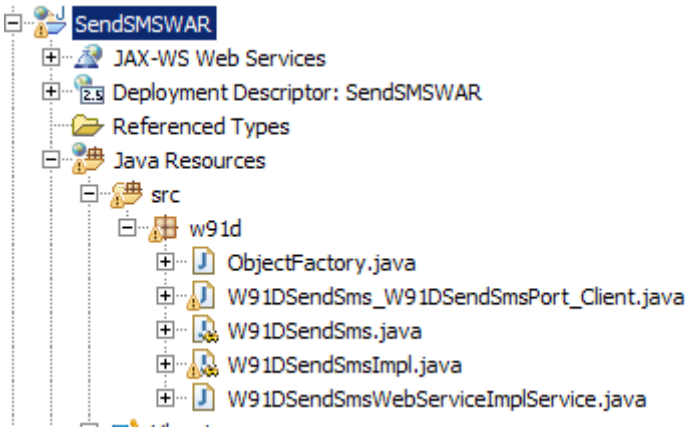


En la configuración “Web Service runtime” seleccionar Apache CXF



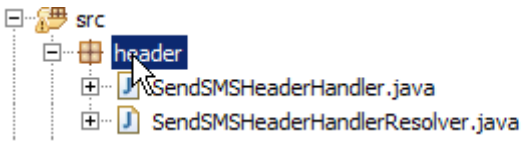
Pulsamos ok y finalizamos el asistente.

Esto creará una serie de ficheros fuente con el cliente web service.



8.3 Creación de Handlers

Crearemos un handler para inyectar la autenticación en la cabecera.



Los fuentes son:

SendSMSHeaderHandler.java

```
package header;

import java.io.ByteArrayInputStream;
import java.util.Set;


import javax.xml.namespace.QName;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPHeader;
import javax.xml.transform.Result;
import javax.xml.transform.Source;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMResult;
import javax.xml.transform.dom.DOMSource;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;

import org.w3c.dom.Document;

public class SendSMSHeaderHandler implements SOAPHandler<SOAPMessageContext> {

	private static final String XML_VERSION_ENCODING = "<?xml version='1.0'
encoding='ISO-8859-1'?>";
	private static final String IDENTITY_HEADER_NAMESPACE_URI =
"com/ejie/documents/xml";
	private static final String IDENTITY_HEADER_LOCAL_PART = "w91dauth";
	private static final String IDENTITY_HEADER_NAMESPACE_PREFIX = "w91d";

	private TransformerFactory transformerFactory;
	private String securityToken = null;

	public SendSMSHeaderHandler(String securityToken) throws Exception {
		this.securityToken = securityToken;
		this.transformerFactory = TransformerFactory.newInstance();
		if (securityToken != null && !securityToken.isEmpty()) {
			securityToken = securityToken.replaceAll(XML_VERSION_ENCODING,
"");
		} else {
			throw new Exception("Error: security token cannot be null or
empty.");
		}
	}

	public boolean handleMessage(SOAPMessageContext context) {
		boolean bValid = false;
		Boolean outboundProperty = (Boolean)
context.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
		if (outboundProperty.booleanValue()) {
			try {
				// Insercion del token de XLNets
				DocumentBuilder docBuilder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
				Document token = docBuilder.parse(new
ByteArrayInputStream(securityToken.getBytes()));
				// Recuperar cabecera SOAP
				SOAPHeader soapHeader =
context.getMessage().getSOAPHeader();
				// Anyadir un child element de tipo w43sIdentity. Es
necesario ya que el elemento debe tener namespace, y si se inserta
				// directamente el XML de N38 no tiene namespace.
				SOAPElement soapElement = soapHeader.addChildElement(new
QName(IDENTITY_HEADER_NAMESPACE_URI, IDENTITY_HEADER_LOCAL_PART,
IDENTITY_HEADER_NAMESPACE_PREFIX));
				// En SOAP 1.2 la cabecera debe tener namespace
```

```
                        // Incluir token en la cabecera creada
                        Source source = new DOMSource(token);
                        Result result = new DOMResult(soapElement);

                        this.transformerFactory.newTransformer().transform(source,
result);
                        bValid = true;
                    } catch (Exception e) {
                        e.printStackTrace();
                        bValid = false;
                    }
                } else {
                    // Nothing to do..
                    bValid = false;
                }
                return bValid;
        }

    public Set<QName> getHeaders() {
        //throw new UnsupportedOperationException("Not supported yet.");
        return null;
    }

    public boolean handleFault(SOAPMessageContext context) {
        //throw new UnsupportedOperationException("Not supported yet.");
        return true;
    }

    public void close(MessageContext context) {
    //throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

SendSMSHeaderHandlerResolver.java

```
package header;

import java.util.ArrayList;
import java.util.List;
import javax.xml.ws.handler.Handler;
import javax.xml.ws.handler.HandlerResolver;
import javax.xml.ws.handler.PortInfo;

public class SendSMSHeaderHandlerResolver implements HandlerResolver {

        private String securityToken = null;

        public SendSMSHeaderHandlerResolver(String securityToken) {
                this.securityToken = securityToken;
        }

        @SuppressWarnings("rawtypes")
        public List<Handler> getHandlerChain(PortInfo portInfo) {
                List<Handler> handlerChain = new ArrayList<Handler>();
                try{
                        SendSMSHeaderHandler headerHandler = new
SendSMSHeaderHandler(securityToken);
                        handlerChain.add(headerHandler);
                } catch (Exception e) {
                        // TODO Tratar la excepcion
                        e.printStackTrace();
                }
                return handlerChain;
        }
}
```

8.4 Uso del cliente

En el apartado anterior se ha generado un fuente java **W91DSendSms_W91DSendSmsPort_Client.java** en el que hay un ejemplo de invocación.

Introduciremos la configuración del handler con la autenticación y un mensaje de prueba básico como en el apartado 4.1

```
public static void main(String args[]) throws java.lang.Exception {
        URL wsdlURL = W91DSendSmsWebServiceImplService.WSDL_LOCATION;

        W91DSendSmsWebServiceImplService ss = new
W91DSendSmsWebServiceImplService(wsdlURL, SERVICE_NAME);
        String strToken =
             "<authenticationLatinia>"+
                   "<userLatinia>innovus.superusuario</userLatinia>"+
                   "<passwordLatinia>MARKSTAT</passwordLatinia>"+
                   "<refProduct>W91DTEST</refProduct>"+
                   "<idContract>2022</idContract>"+
                   "<loginEnterprise>INNOVUS</loginEnterprise>"+
                   "<password>xxxxxx</password>"+
              "</authenticationLatinia>";
        SendSMSHeaderHandlerResolver headerHandlerResolver = new
SendSMSHeaderHandlerResolver(strToken);
        ss.setHandlerResolver(headerHandlerResolver);
        W91DSendSms port = ss.getW91DSendSmsPort();

        System.out.println("Invoking sendSms...");
        //java.lang.String _sendSms_stringInput = "_sendSms_stringInput1535251077";
        String _sendSms_stringInput="<PETICION><LATINIA><MENSAJES><MENSAJE_INFO
ACUSE=\"S\"><GSM_DEST>688672904</GSM_DEST><TEXTO>Hola mundo</TEXTO>"+
             "</MENSAJE_INFO></MENSAJES></LATINIA></PETICION>";
        java.lang.String _sendSms__return = port.sendSms(_sendSms_stringInput);
        System.out.println("sendSms.result=" + _sendSms__return);

}
```

8.5 Uso del cliente en diferentes entornos

Hay que tener en cuenta en el ejemplo hemos creado el cliente con un WSDL de desarrllo, por lo que es éste el que se usa por defecto, pero en los diferentes entornos el WSDL es diferente, por lo que en la invocación del cliente hay que indicar el WSDL con la URL que corresponda a cada entorno. El SERVICE_NAME no varía

Ejemplo para producción intranet:

```
URL wsdlURL = new
URL("http://svc.intra.integracion.jakina.ejgvdns/ctxapp/W91dSendSms?wsdl");
QName SERVICE_NAME = new QName("http://w91d","W91dSendSmsWebServiceImplService");
W91DSendSmsWebServiceImplService ss = new W91DSendSmsWebServiceImplService(wsdlURL,
SERVICE_NAME);
```

9 ANEXO 2. Creación de MDB para tratamiento de eventos y mensajes

9.1 Crear cola JMS

Primero se deberá disponer de una cola propia en la que el simularemos el envío de mensajes por parte del propagador

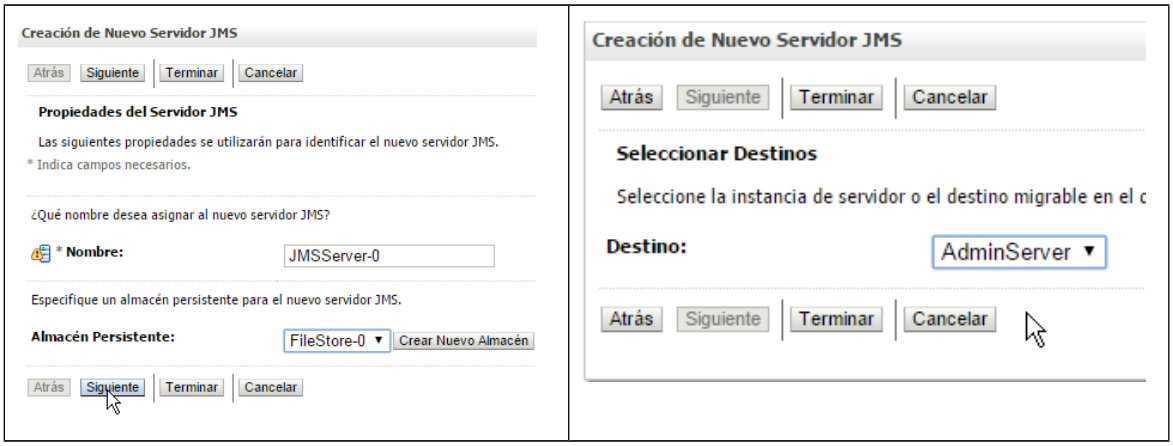
A continuación crearemos una cola mediante la consola de Weblogic11 en nuestra instancia local.

Si ya se dispone de un servidor y un módulo JMS pueden saltarse los dos primeros pasos y crear la cola directamente sobre el módulo existente.

9.1.1. Crear el servidor JMS

Desde la consola, seleccionar la opción Servicios->Mensajes->Servidores JMS y pulsar el botón “Nuevo”

Para simplificar el ejemplo pondremos un almacenamiento en fichero, si no disponemos de uno, crearemos uno nuevo “FileStore-0”



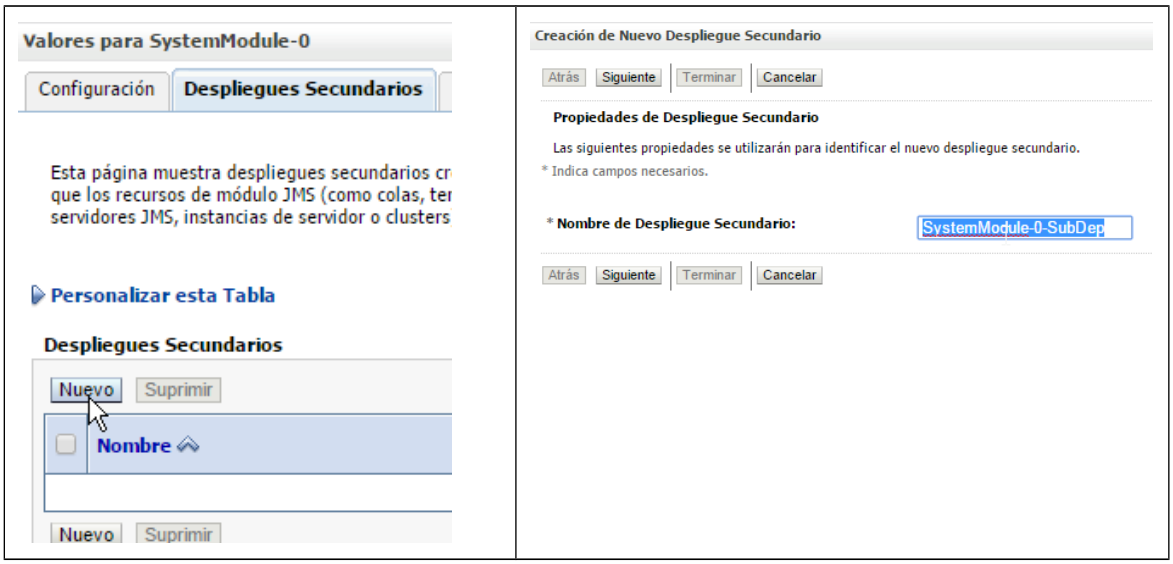
Para terminar indicaremos nuestro servidar local como destino.

9.1.2. Crear el Módulo JMS

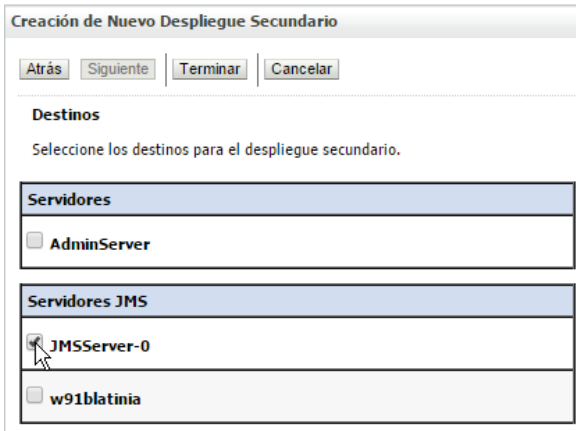
Desde la consola, seleccionar la opción Servicios->Mensajes->Módulos JMS y pulsar el botón “Nuevo”

Crearemos un nuevo módulo “SystemModule-0”

Posteriormete crear el despliegue secundario “**SystemModule-0-SubDep**”

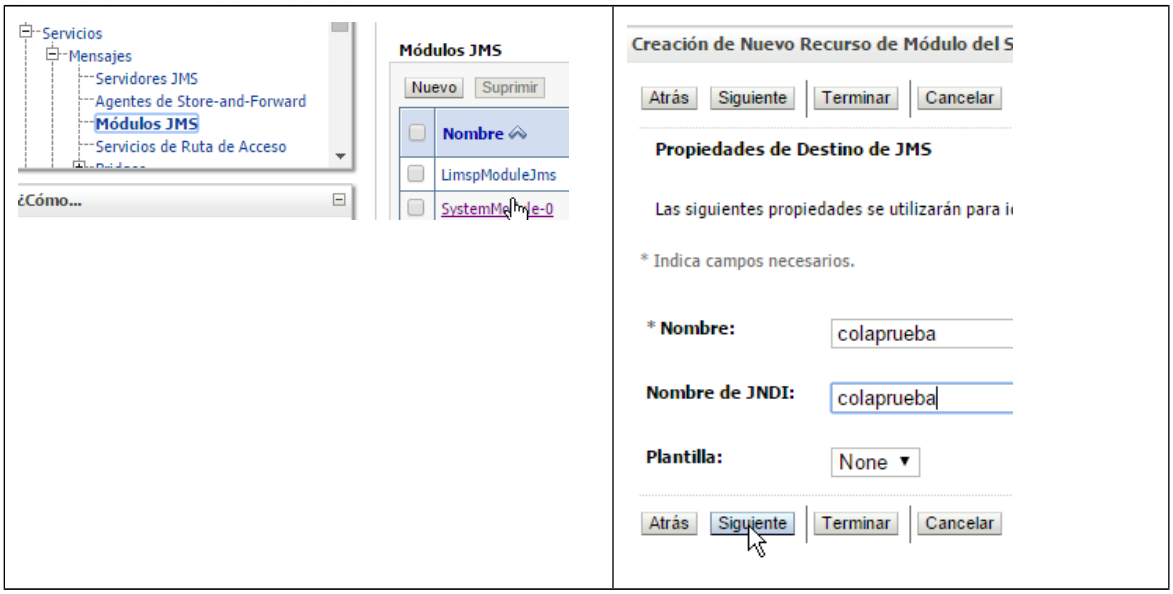


En el siguiente paso desplegar el módulo en el Servidor JMS creado anteriormente

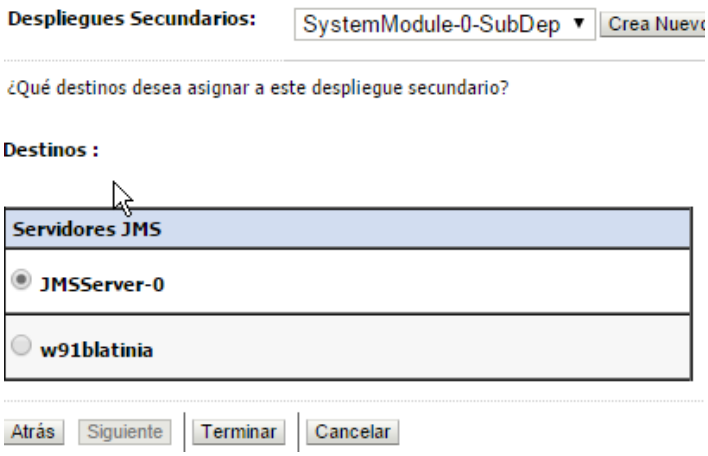


9.1.3. Crear la cola

Una vez creado el módulo JMS, lo seleccionaremos y crearemos una nueva cola “colaprueba”

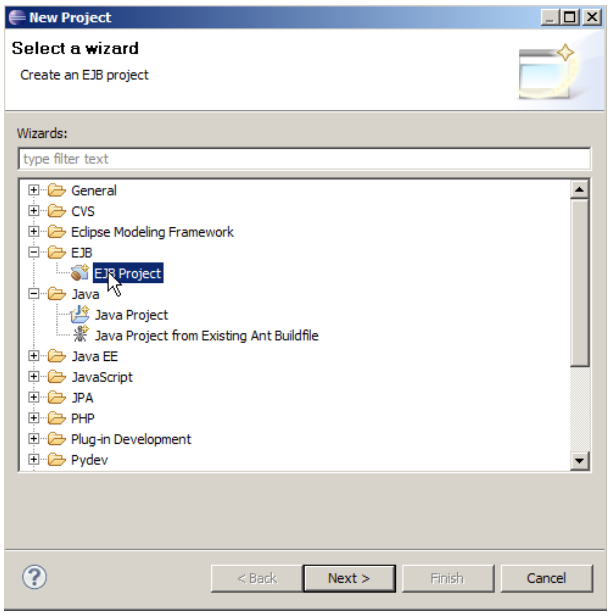


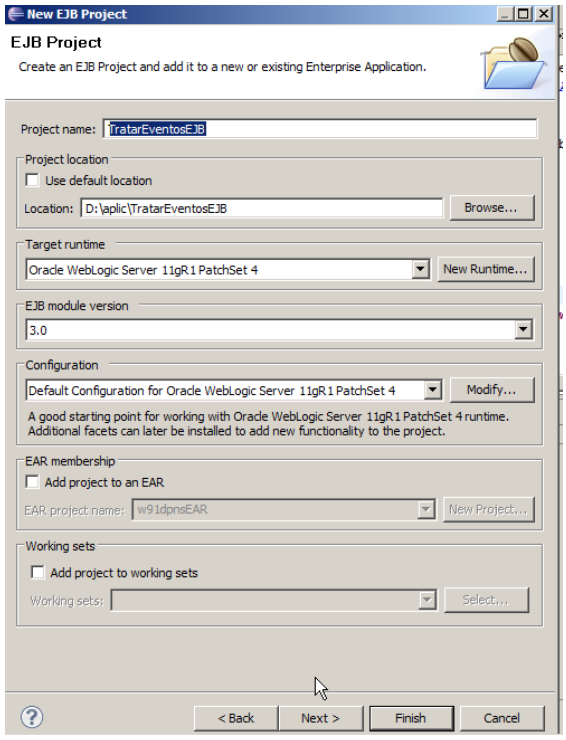
En el siguiente paso seleccionaremos el espliegue secundario creado anteriormente con el servidor JMS y finalizamos la creación de la cola JMS



9.2 Crear el proyecto EJB

Crearemos un nuevo proyecto EJB **TratarMensajesEJB** y dentro un nuevo "**Message Driven Bean**" **(**MDB) que se conectará a la cola recién creada

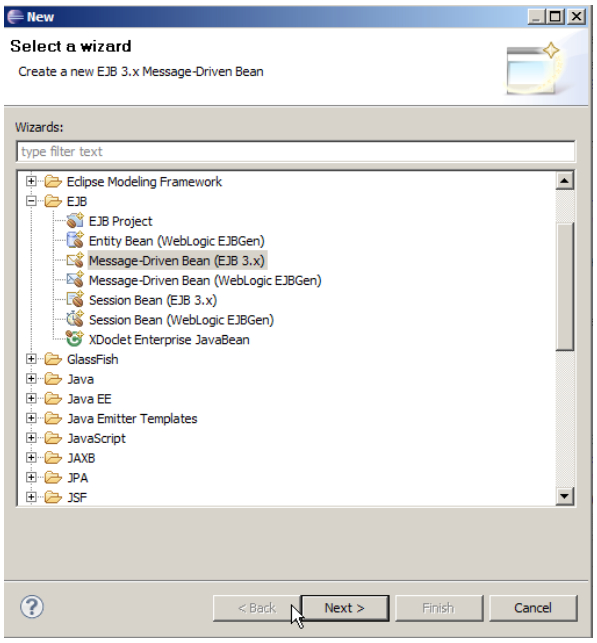


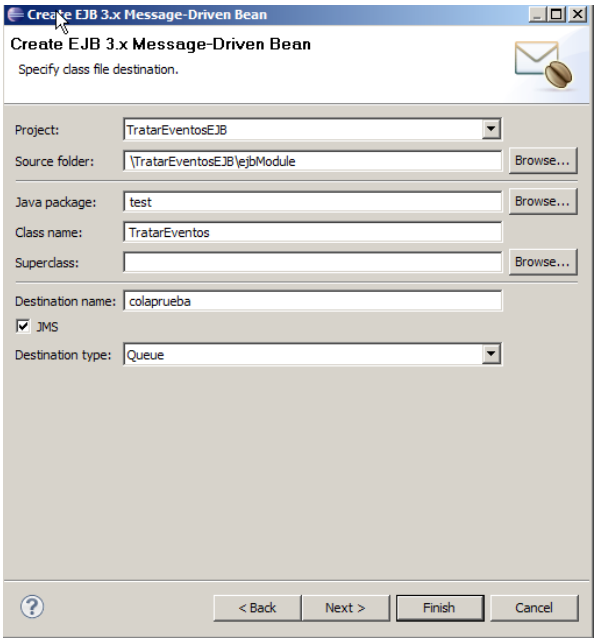


En el asistente activaremos la opción de crear el descriptor ejb-jar.xml

9.3 Crear el MDB (Message Driven Bean)

Crearemos un MDB que consumirá los mensajes que el propagador/enrutador envie a la cola creada en los pasos anteriores **"colaprueba".** En este caso craremos una clase **"TratarEventos"**





El asistente generará la clase **TratarEventos** que se conectará a la cola indicada. El método "**onMessage"** se ejecutará cuando lleguen mensajes a la cola. Este es el método que se debe adaptar a las necesidades del negocio.

En el ejemplo usaremos el mismo MDB para tratar tanto los eventos como los mensajes del usuario, por lo lo que se discriminará en el propio código cuándo se recibe uno u otro.

```
package test;

import java.io.StringReader;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
import javax.xml.XMLConstants;
import javax.xml.namespace.NamespaceContext;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;

/**
 * Message-Driven Bean implementation class for: TratarEventos
 *
 */
@MessageDriven(
             activationConfig = { @ActivationConfigProperty(
                          propertyName = "destinationType", propertyValue =
"javax.jms.Queue"
             ) },
             mappedName = "colaprueba")
public class TratarEventos implements MessageListener {

    /**
     * Default constructor.
     */
    public TratarEventos() {
        // TODO Auto-generated constructor stub
    }

      /**
     * @see MessageListener#onMessage(Message)
     */
    public void onMessage(Message message) {
      try {
                  if (message instanceof TextMessage) {
                         // SE CASTEA EL TIPO DE MENSAJE DE COLA
                         TextMessage textMessage = (TextMessage) message;
                         String xmlMsg=textMessage.getText();
                         System.out.println("textMessage.getText(): "+xmlMsg);

                         Map<String, String> propiedades=null;
                         // Testear el tipo de mensaje recibido
                         String what = textMessage.getStringProperty("EventWhat");

                         if("LATINIA_ESTADO".equalsIgnoreCase(what)){
                                System.out.println("Se ha recibido un estado");
                                propiedades=leerNotificationBody("EVENT",xmlMsg);
```

```
                        }else if("LATINIA_MENSAJE".equalsIgnoreCase(what)){
                                System.out.println("Se ha recibido un mensaje");
                                propiedades=leerNotificationBody("MENSAJE",xmlMsg);
                        }else{
                                System.out.println("Tipo de mensaje desconocido: "+
what);
                        }
                        if(null!=propiedades){
                                for (Map.Entry<String, String> entry :
propiedades.entrySet()) {
                                        System.out.println(entry.getKey() + " : " +
entry.getValue());
                                }
                        }

                } else {
                        System.out.println("Tipo de mensaje JMS invalido");
                }
            } catch (Exception e) {
                e.printStackTrace();
            }

    }

    private Map<String, String> ParsearXMLValue(String Tipo,Map<String, String>
propiedades) throws XPathExpressionException {
       String xmlValue=propiedades.get("xmlValue");

       XPathFactory xpFactory = XPathFactory.newInstance();
             XPath xpath = xpFactory.newXPath();
             XPathExpression propertyExpression = xpath.compile("/"+Tipo);

       InputSource in = new InputSource(new StringReader(xmlValue));
             NodeList nodes = (NodeList) propertyExpression.evaluate(in,
XPathConstants.NODESET);
             //Sacar los valores
             nodes = nodes.item(0).getChildNodes();
             String nombre;
             String valor;
             for (int i = 0; i < nodes.getLength(); i++) {
                        Node node = (Node) nodes.item(i);
                        nombre=node.getNodeName();
                        if("ESTADOS".equalsIgnoreCase(nombre)){
                                NodeList estados=node.getChildNodes();
                                for (int e = 0; e < estados.getLength(); e++) {
                                        Node estado=estados.item(e);
                                        nombre=estado.getNodeName();
                                        valor=estado.getFirstChild().getNodeValue();

                                        propiedades.put(nombre, valor);
                                }
                        }else{
                                valor=node.getFirstChild().getNodeValue();
                                propiedades.put(nombre, valor);
                        }

             }
             return propiedades;

    }

    private  Map<String,String> leerNotificationBody(String Tipo,String xml ) throws
XPathExpressionException {
       Map<String, String> propiedades = new HashMap<String, String>();
       //            CONFIGURACION DEL XPATH
             XPathFactory xpFactory = XPathFactory.newInstance();
             XPath xpath = xpFactory.newXPath();
             xpath.setNamespaceContext(new PropiedadesNamespaceContext());
```

```
            XPathExpression propertyExpression = xpath.compile("//ns:Property");
            XPathExpression propertyNameExpression =
xpath.compile("./ns:Name/text()");
            XPathExpression propertyValueExpression =
xpath.compile("./ns:Value/text()");
            XPathExpression xmlExpression = xpath.compile("//ns:xmlValue");

            // OBTENCION DEL XML
            InputSource in = new InputSource(new StringReader(xml));
            NodeList nodes = (NodeList) propertyExpression.evaluate(in,
XPathConstants.NODESET);

            // BUSQUEDAS XPATH Y CARGA DEL HASHMAP QUE SE DEVUELVE
            for (int i = 0; i < nodes.getLength(); i++) {
                  Node node = (Node) nodes.item(i);
                  String name = (String) propertyNameExpression.evaluate(node,
XPathConstants.STRING);
                  String value = (String) propertyValueExpression.evaluate(node,
XPathConstants.STRING);
                  propiedades.put(name, value);
            }
            // OBTENCION DEL XML QUE VIENE DENTRO DEL XML

            //Obtencion del XML que viene dentro del XML
            in = new InputSource(new StringReader(xml));

            String xmlValue = (String) xmlExpression.evaluate(in,
XPathConstants.STRING);
            propiedades.put("xmlValue", xmlValue);
            return ParsearXMLValue(Tipo,propiedades);
      }

      /**
          * @author ERPI
          * Clase necesaria en xpath para xml con namespaces
          *
          */
         private class PropiedadesNamespaceContext implements NamespaceContext {
            final String prefijo = "ns";
            final String namespace = "com/ejie/notification/xml";

            public String getNamespaceURI(String prefix) {
                  if (prefix == null)
                        throw new NullPointerException("Null prefix");
                  else if (prefijo.equals(prefix))
                        return namespace;
                  else if ("xml".equals(prefix))
                        return XMLConstants.XML_NS_URI;
                  return XMLConstants.NULL_NS_URI;
            }

            // This method isn't necessary for XPath processing.
            public String getPrefix(String uri) {
                  throw new UnsupportedOperationException();
            }

            // This method isn't necessary for XPath processing either.
            public Iterator<String> getPrefixes(String uri) {
                  throw new UnsupportedOperationException();
            }
         }
}
```

9.4 Probar el MDB

Es recomendable probar el MDB en local para aprovechar las facilidades que ofrecen los IDE de desarrollo.

En este caso utilizaremos SoapUI para inyectar los mensajes y HermesJMS para realziar la conexión a nuestra cola en local.

En la web de SoapUI http://www.soapui.org/ está disponible el software y la documentación sobre cómo configurar las llamadas JMS. En el ejemplo utilizaremos Weblogic que no aparece en la documentación por lo que la configuración de **HermesJMS** para usar nuestra cola local puede hacerse de la siguiente manera (tener el weblogic arrancado y la cola configurada)

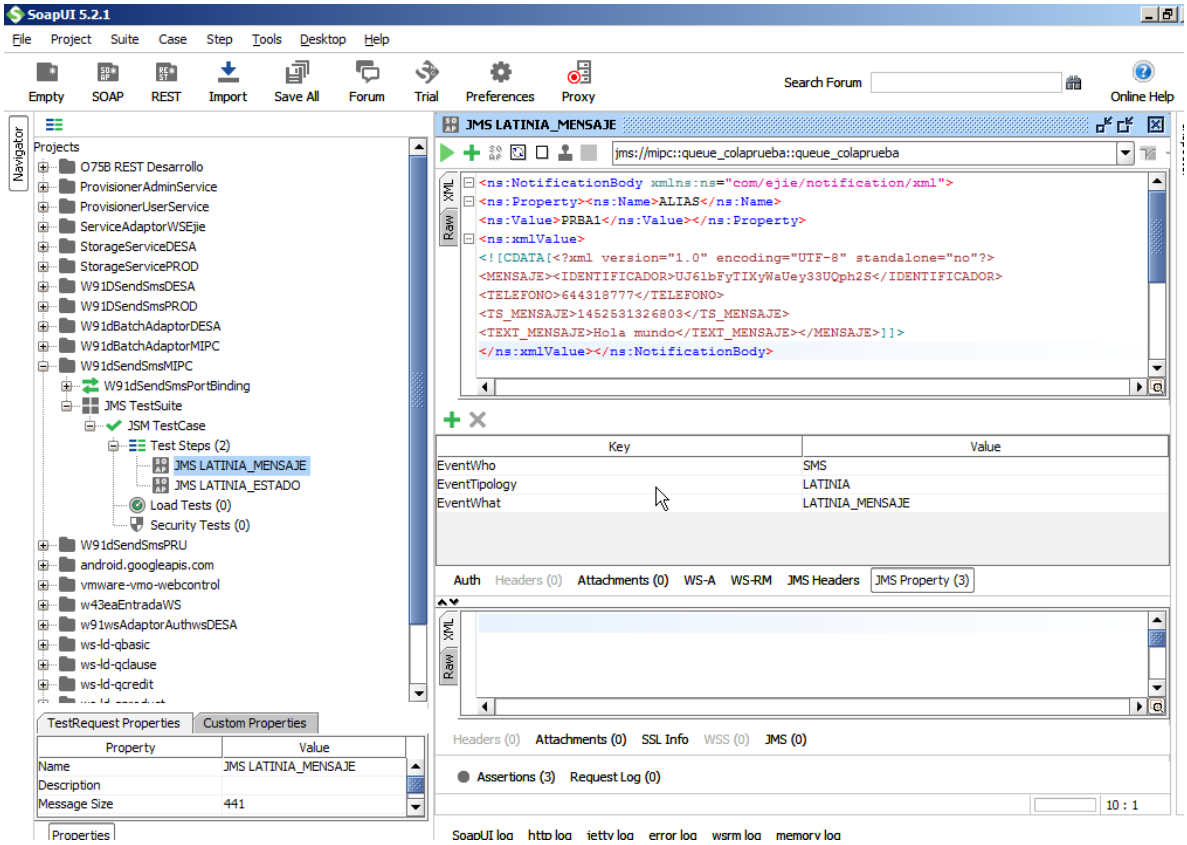
1. **Crear sesión:** Sessions->New->New sesión. Poner un nombre: ejemplo **"mipc"**
2. **Indicar Provider:** Acceder a la zona de proveedores n la solapa inferior "Providers". Una vez en ella con el botón derecho seleccionar **"Add group"** y poner un nombre ej: "Weblogic10". Una vez creado el grupo, en la zona "Library" con el botón derecho seleccionar la opción **"Add JAR(s)"** y añadir la librería **wlfullclient.jar.** Esta librería está disponible en la zona de documentación de Latinia en http://flexcloud.ejiedes.net/
3. **Connection Factory;** Class: hermes.JNDIQueueConnectionFactory, Loader: Weblogic10 (el provider creado anteriormente)
4. **Connection Factory:** Indicaremos los parámetros de conexión a nuestro weblogic local (botón derecho->Add Propertie)

   Binding: weblogic.jms.ConnectionFactory
   initialContextFactory: weblogic.jndi.WLInitialContextFactory
   providerURL: t3://localhost:7001
   securityCredentials: weblogic11
   securityPrincipal: weblogic11"
5. **Discover:** Una vez realizada la configuración, con la opción Discover (jms->Sessions->mipc Botón derecho Discover) deben aparecer todas las colas definidas en nuestro weblogic entre la que estará la cola para publicar los mensajes.

Una vez creada la conexión a nuestra cola en HermesJMS, puede añadirse el end-point con la cola jms para realizar el envío taly como se describe en la web de SoapUI y activar el debug en el IDE de Eclipse en la función onEvent del MDB.

Es necesario añadir las propiedades JMS para simular el envío que realiza el propagador

EventWho: SMS
EventTipology: LATINIA
EventWhat: LATINIA_MENSAJE (ó LATINIA_EVENTO)


SoapUI 5.2.1
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Search Forum
Online Help
Navigator
Projects
O75B REST Desarrollo
ProvisionerAdminService
ProvisionerUserService
ServiceAdaptorWSEjie
StorageServiceDESA
StorageServicePROD
W91DSendSmsDESA
W91DSendSmsPROD
W91dBatchAdaptorDESA
W91dBatchAdaptorMIPC
W91dSendSmsMIPC
W91dSendSmsPortBinding
JMS TestSuite
JSM TestCase
Test Steps (2)
JMS LATINIA_MENSAJE
JMS LATINIA_ESTADO
Load Tests (0)
Security Tests (0)
W91dSendSmsPRU
android.googleapis.com
vmware-vmo-webcontrol
w43eaEntradaWS
w91wsAdaptorAuthwsDESA
ws-ld-qbasic
ws-ld-qclause
ws-ld-qcredit
TestRequest Properties
Custom Properties
Property
Value
Name
JMS LATINIA_MENSAJE
Description
Message Size
441
Properties
JMS LATINIA_MENSAJE
jms://mipc::queue_colaprueba::queue_colaprueba
XML
Raw
<ns:NotificationBody xmlns:ns="com/ejie/notification/xml">
<ns:Property><ns:Name>ALIAS</ns:Name>
<ns:Value>PRBA1</ns:Value></ns:Property>
<ns:xmlValue>
<![CDATA[<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<MENSAJE><IDENTIFICADOR>UJ61bFyTIXyWaUey33UQph2S</IDENTIFICADOR>
<TELEFONO>644318777</TELEFONO>
<TS_MENSAJE>1452531326803</TS_MENSAJE>
<TEXT_MENSAJE>Hola mundo</TEXT_MENSAJE></MENSAJE>]]>
</ns:xmlValue></ns:NotificationBody>
Key
Value
EventWho
SMS
EventTipology
LATINIA
EventWhat
LATINIA_MENSAJE
Auth Headers (0) Attachments (0) WS-A WS-RM JMS Headers JMS Property (3)
Headers (0) Attachments (0) SSL Info WSS (0) JMS (0)
Assertions (3) Request Log (0)
10 : 1
SoapUI log http log jetty log error log wsrm log memory log