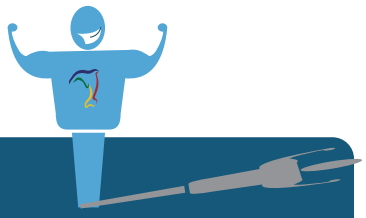
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

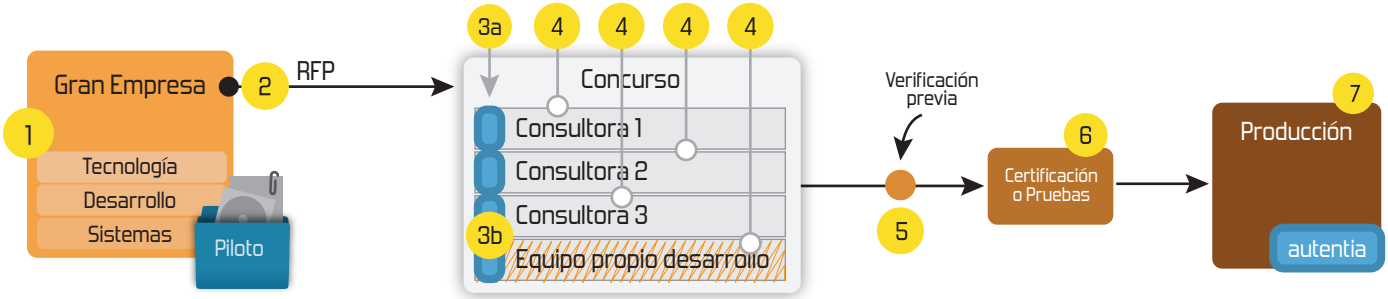
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Juan Diego Reyes Ponce

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero en Informática

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2014-01-23 Tutorial visitado 43 veces Descargar en PDF

Construir un cliente REST con PowerBuilder .NET

Índice de contenidos

1. Introducción

¿Qué es PowerBuilder?

Creado inicialmente por Sybase, y ahora producto de SAP, PowerBuilder es una herramienta para el desarrollo de aplicaciones de alto rendimiento y basadas en modelos de datos, para escritorio o la web. Puede parecer algo anticuado, sabiendo que disponemos de herramientas basadas en soluciones Windows de Microsoft más potentes, y que también puede quedar un poco corta en cuanto al estilo de las interfaces de usuario, pero no deja de ser una herramienta muy potente en cuanto a la optimización de recursos, facilidad y rapidez de desarrollo, e interfaces de usuario sencillas e intuitivas. A título personal, uno de los mayores logros de cualquier herramienta de desarrollo, es hacerle la vida más fácil al desarrollador. Y PowerBuilder lo consigue, ofreciéndote componentes visuales que hacen que ahorres tiempo en diseñar tu aplicación, te ofrece componentes integrados de .NET, librerías precompiladas, etc.

Una de las suites que ofrece PowerBuilder es PowerBuilder .NET, que tiene un triple objetivo:

- Simplificar el desarrollo de aplicaciones en .NET
- El desarrollo de productos que permitan a los clientes convertir sus aplicaciones de Win32 a .NET (WPF) para aplicaciones de escritorio, obteniendo todas las ventajas que ofrece
- posibilidad de creación de aplicaciones con WCF (aplicaciones web, servicios web, etc.).

Para más información sobre WPF y WCF, consulta el apartado de Referencias.

Powerscript

La base del éxito de PowerBuilder es su lenguaje nativo orientado a objetos: Powerscript. Dicho lenguaje (cuya documentación técnica tenéis aquí) está optimizado (mediante su elemento estrella: DataWindow) para poder realizar operaciones de conexión a base de datos, consultar y recorrer dataSets de forma eficiente, etc. Permite también acceder a todas las librerías .NET, por lo que puedes obtener todas las ventajas que ofrecen las mismas, haciendo casi lo mismo que podrías hacer con otro lenguaje, como C# o VB.NET. Con Powerscript también puedes cargar cualquier librería DLL.
Por ejemplo, tenemos el siguiente código en C#, que realiza un postprocesamiento de pintado de filas de un grid de forma manual:

```
view plain print ?
01. private void myGrid_RowPostPaint(object sender, DataGridViewRowPostPaintEventArgs e)
02. {
```

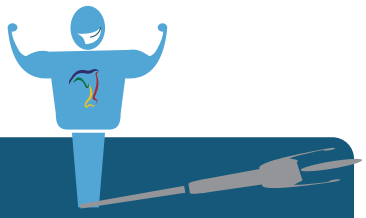
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

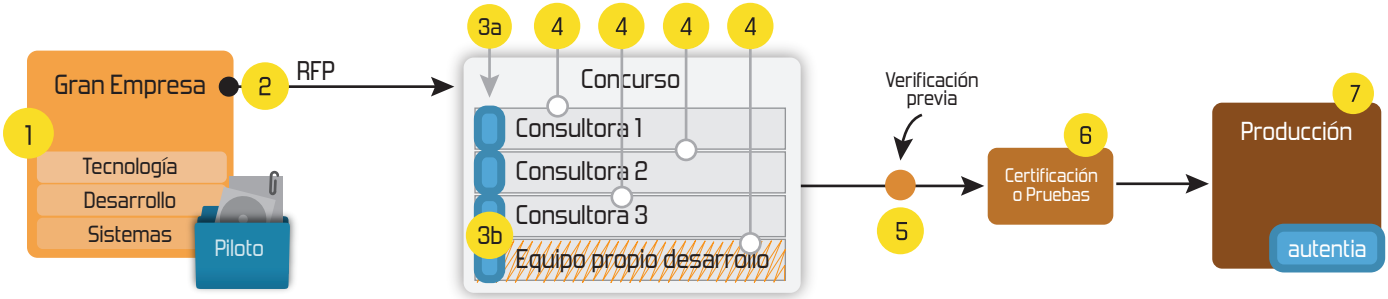
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

```
    if (e.RowIndex == 2)
    {
        int rowHeaderWidth = myGrid.RowHeadersVisible
        myGrid.RowHeadersWidth : 0;
        Rectangle rowBounds = new Rectangle(
                                        myGrid.RowHeadersWidth,
                                        e.RowBounds.Top,
                                        myGrid.Columns.GetColumnsWidth(DataGridViewElementStates.Vis
                                        e.RowBounds.Height);
        ControlPaint.DrawBorder(e.Graphics, rowBounds,
        Color.Red, ButtonBorderStyle.Solid);
        myGrid.Rows[e.RowIndex].DefaultCellStyle.BackColor =
        Color.BlanchedAlmond;
    }
}
```

En Powerscript:

```
view plain  print  ?
This.SelectRow(row, true)
```

PowerBuilder ya se encarga de definir el elemento *row*, asignándole las propiedades de forma y color de forma automática con una serie de valores por defecto, y deja al desarrollador la libertad de poder modificar dichas propiedades con una interfaz intuitiva, que recuerda a la suite de Visual Studio .NET (de hecho la versión de PowerBuilder .NET está basada en dicha suite).

PowerBuilder trabaja con dos tipos de librerías: Librerías PBL (pseudocódigo) y librerías DLL (código máquina), lo que permite la creación de aplicaciones modulares con la flexibilidad del lenguaje descrito arriba. Es importante hacer notar que no son compatibles entre sí, es decir, que a la hora de generar un ejecutable, tendremos que dar a elegir a PowerBuilder si generar las librerías en formato PBL o DLL.

Objetivo del tutorial

En este tutorial vamos a crear un sencillo cliente REST con PowerBuilder .NET, que consuma un servicio ya desplegado en la web, y devuelva un resultado en formato JSON (también podría soportar XML). Veréis de qué forma tan sencilla podremos generar el cliente (la propia suite nos ofrece wizards) y, con unas líneas de Powerscript, hacer la llamada, y devolver el resultado.
El servicio que consumirá nuestro cliente será el API de direcciones de Google (https://developers.google.com/maps/documentation/directions/), para obtener la ruta en coche a seguir entre dos direcciones.

2. Entorno

Para la realización de este tutorial, se ha usado el siguiente entorno:

- Macbook pro core i7 con 16gb RAM
- SO: Mavericks
- Virtualbox 4.3.4 con máquina virtual Windows XP SP3 (descarga)
- Microsoft Windows SDK for Windows 7 and .NET Framework 4 (descarga)
- .NET Framework 4.0 (descarga)
- IDE: PowerBuilder .NET 12.5.2 (descarga)

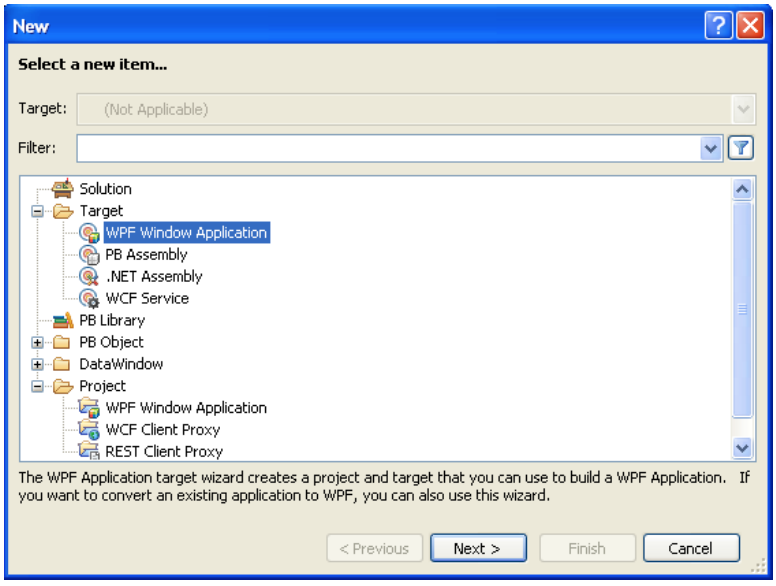
Los productos que incluye la distribución de PowerBuilder v12.5.2 son:

- Visual Studio 10 Service Pack 1 (Console)
- PowerBuilder 12.5.2
- PowerBuilder .NET 12.5.2 basado en Visual Studio 10 (requiere Microsoft Windows SDK .NET Framework 4.0)
- Sql Anywhere 12: Servidor de base de datos, permitiendo el desarrollo ágil y el despliegue de aplicaciones que usan bases de datos, siendo dichas aplicaciones de escritorio, móviles y web.
- Infomaker 12.5.2: Generador de informes

3. Creando el proyecto

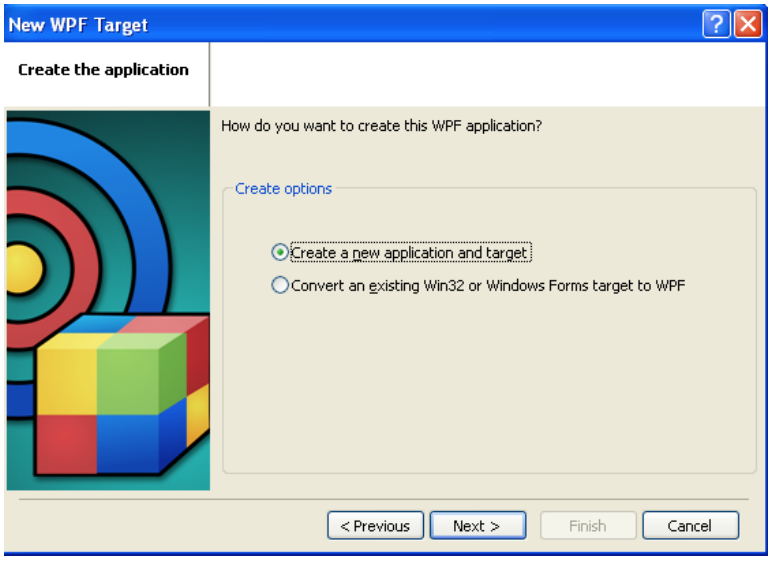
No es objeto de este tutorial explicar todos los conceptos básicos de PowerBuilder: Solution, project, target, library, application, Window, DataWindow, etc. Si desconoces estos elementos, te recomiendo que comiences por aquí antes de seguir con el tutorial.

Para la creación de cualquier proyecto en PowerBuilder, es necesario que esté envuelto en un target (y a su vez en un solution). En nuestro caso, vamos a crear un target WPF Windows application:

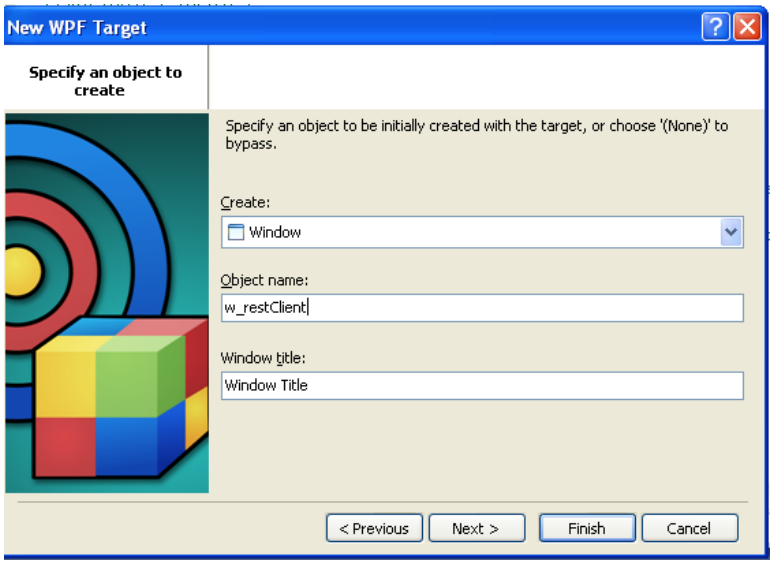


» Spring BeanPostProcessor

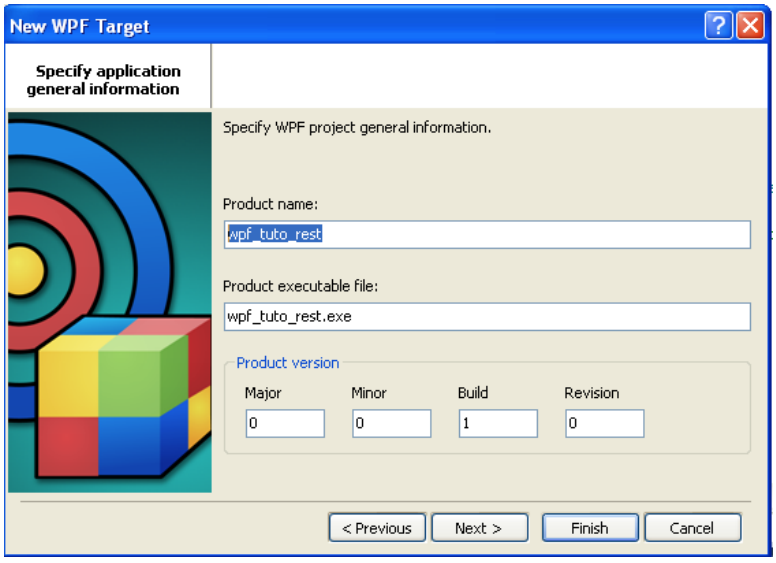
Cambiamos el nombre de la aplicación a *wpf_tuto_rest*



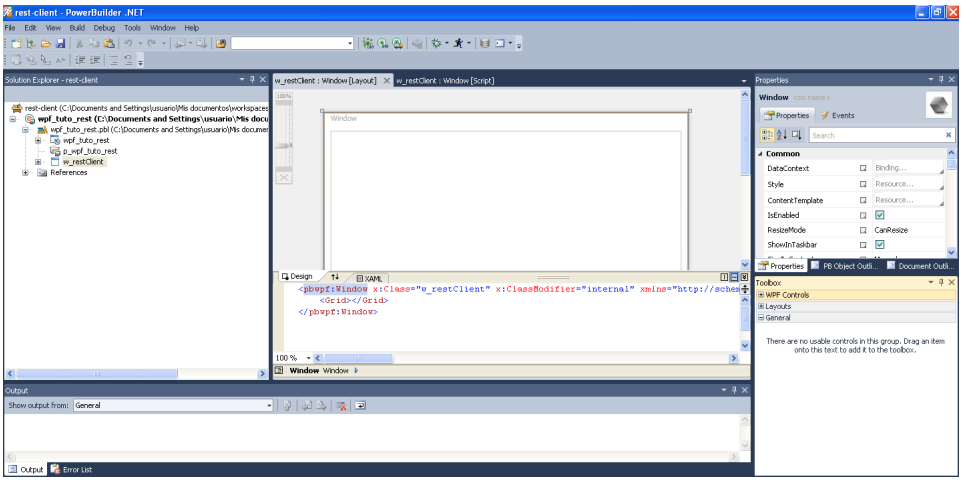
Crearemos un elemento Window, que será la ventana inicial que se abrirá con la ejecución de nuestra aplicación (la notación w_* es para 'window'):



Establecemos el layout 'Grid' por defecto, el nombre del proyecto *p_wpf_tuto_rest*, y el nombre del producto (que coincide con el nombre del ejecutable) *wpf_tuto_rest*, con la versión 0.0.1.0 del producto:



El entorno, y la estructura del árbol del proyecto quedará de la siguiente forma:



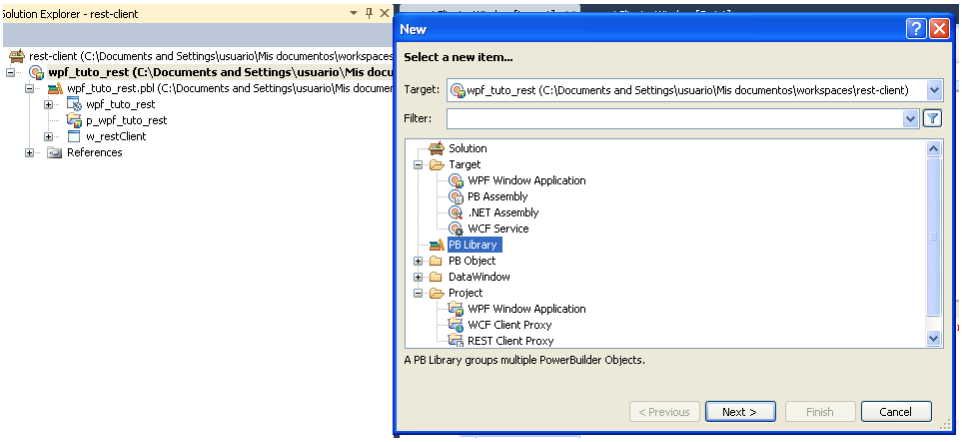
Como vemos, en este momento tenemos creado el target, la aplicación y la ventana, que por ahora no tiene ningún elemento.

4. Creando la librería con el cliente

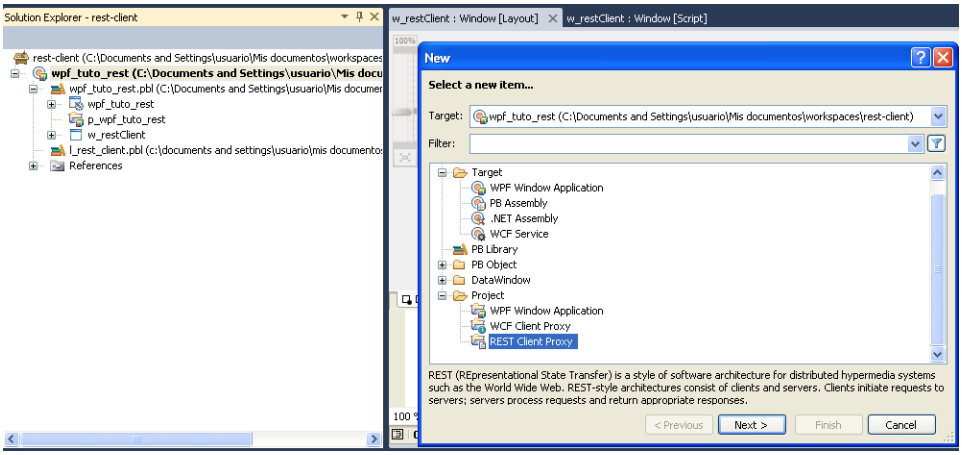
Ya con la aplicación y el target creados, es momento de definir nuestro cliente REST.

Veremos a continuación que en la creación del cliente, PowerBuilder generará una serie de elementos por defecto, y nosotros generaremos un objeto proxy que atacará al servicio, y que podremos usar en cualquier parte de nuestra aplicación.

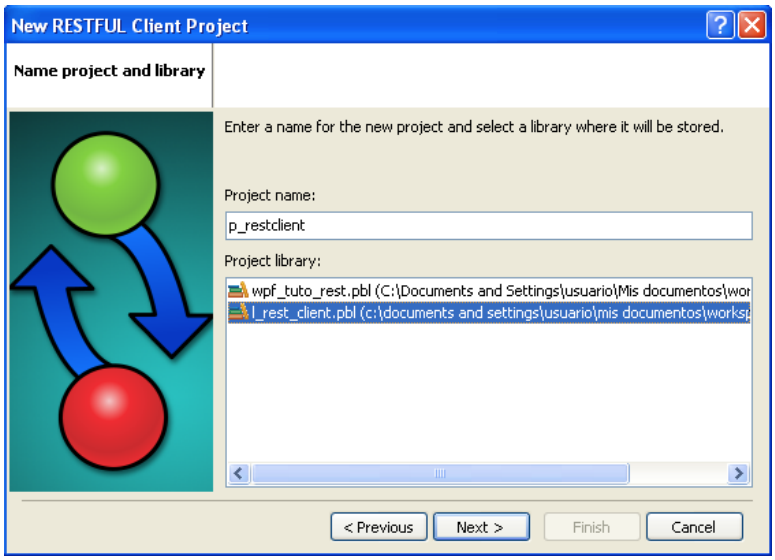
De momento vamos a modularizar nuestra aplicación generando el cliente REST en una librería PBL aparte, que la crearemos seleccionando nuestro target, haciendo clic con el botón derecho sobre él, y seleccionando 'New...' -> 'PB Library'. Le asignaremos el nombre 'l_rest_client':



A continuación, vamos a crear nuestro cliente REST, haciendo clic con el botón derecho al target, 'New...' -> 'Rest Client Proxy'. Esto abrirá el wizard de creación del cliente REST:



En la primera pantalla, estableceremos el nombre por defecto de la aplicación. En la siguiente pantalla, asignaremos la librería recién creada como almacén destino de la aplicación:



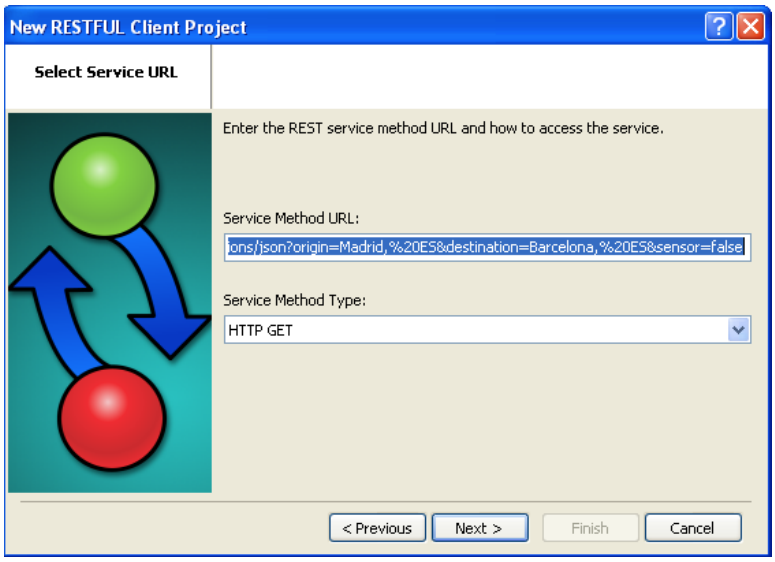
A continuación, nos pedirá la URL del método del servicio. En nuestro caso haremos uso del servicio web de direcciones de Google para obtener las direcciones a seguir para ir de Madrid a Barcelona:

http://maps.googleapis.com/maps/api/directions/json?origin=Madrid,%20ES&destination=Barcelona,%20ES&sensor=false

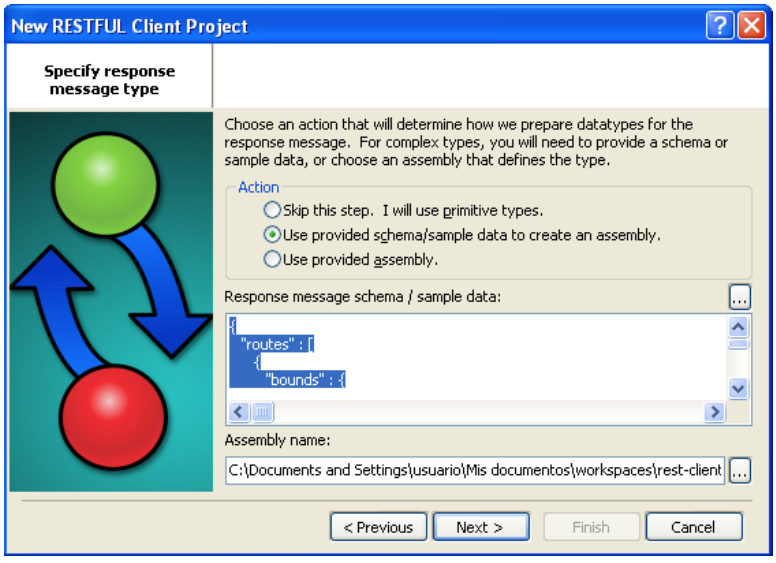
Obviamente, la funcionalidad de nuestra aplicación va a ser bastante limitada, ya que hará siempre la misma consulta. Una mejora (que podríais aplicar como ejercicio) podría ser meter argumentos variables a la llamada (que se representan con {}), por ejemplo:

http://maps.googleapis.com/maps/api/directions/json?origin={origen},%20ES&destination={destino},%20ES&sensor=false

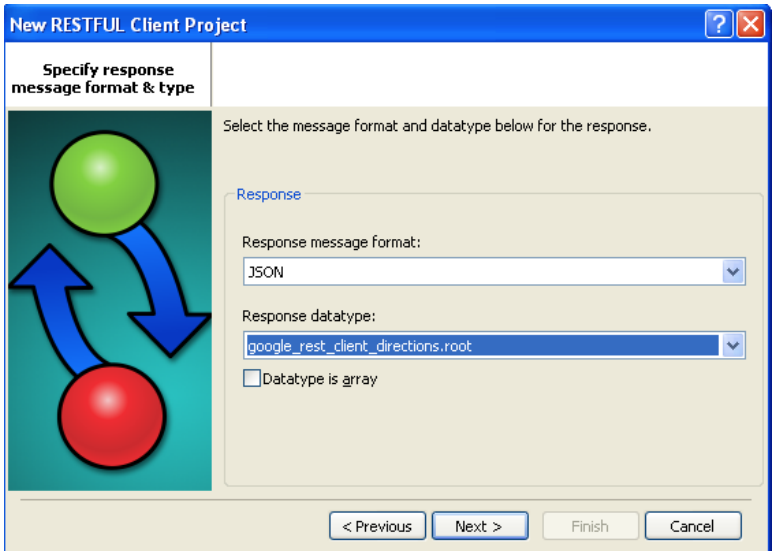
Las variables *origen* y *destino* se mapearán como argumentos al método GetMessage(...) en la generación de nuestro proxy (tened esto en cuenta ahora aunque no lo entendáis. Más adelante explicaremos el objeto proxy).



Lo siguiente que nos pedirá será el esquema de respuesta, para que PowerBuilder pueda generar y mapear los elementos del JSON en objetos. En nuestro caso le indicaremos un esquema por defecto que será la respuesta de la consulta anterior:

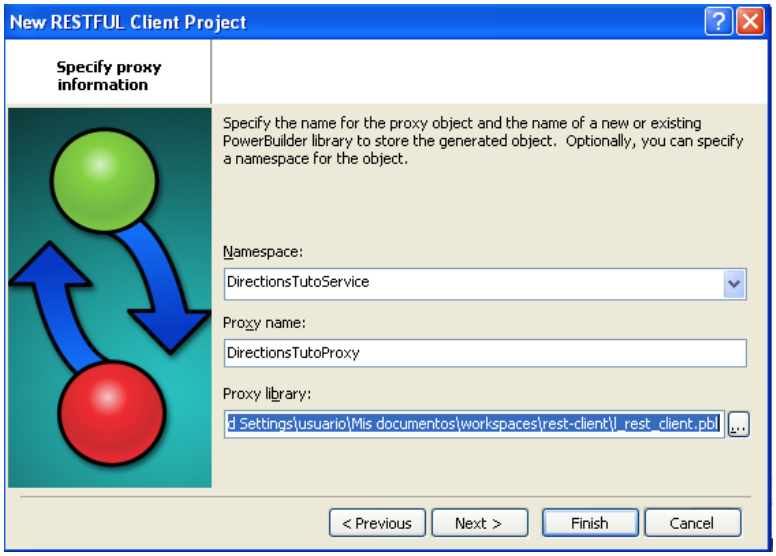


En el siguiente paso le indicaremos el tipo de respuesta y el elemento root:

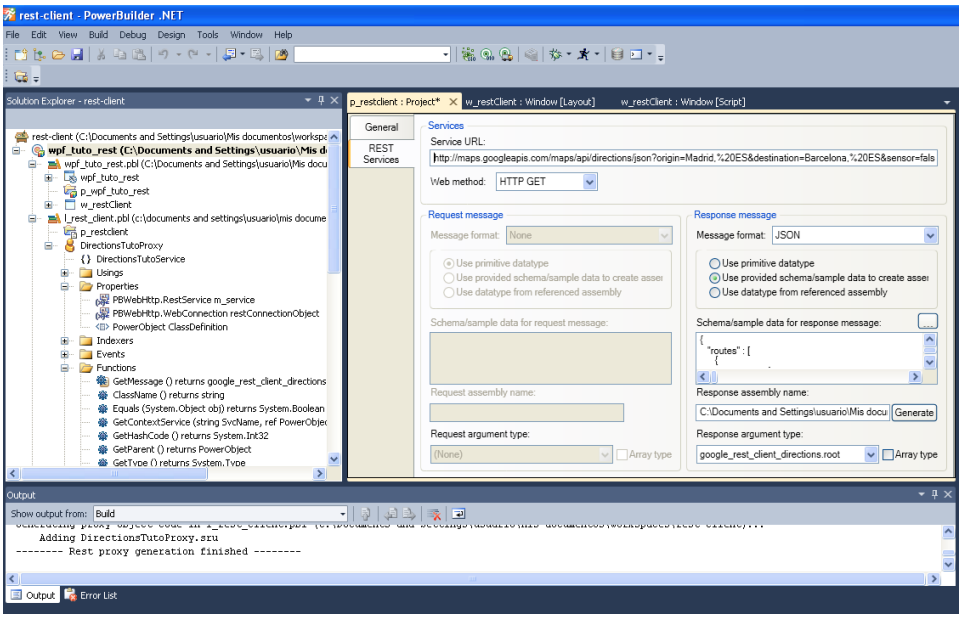


A continuación nos pide los siguientes datos sobre el proxy:

- Namespace: Dato opcional que indicará el namespace del objeto proxy. Le asignaremos el valor *DirectionsTutoService*
- Proxy name: Nombre que tendrá nuestro objeto proxy. Le asignaremos el valor *DirectionsTutoProxy*
- Proxy library: Librería donde se va a desplegar el proxy. Haremos clic en '...' y seleccionaremos *l_rest_client.pbl*. Esto generará el proxy en dicha librería, y dispondremos de dicho proxy desde cualquier aplicación PowerBuilder que incluya esta librería.

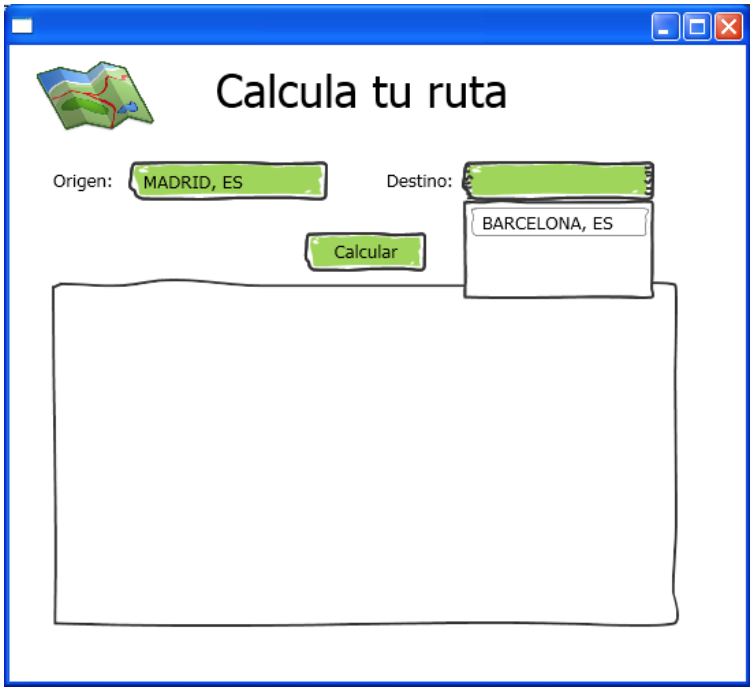


Una vez que tengamos generado el cliente, necesitaremos generar el proxy. Para ello, haremos clic sobre el target en el árbol de proyecto, y haremos clic sobre el icono de generar proxy:  Esto generará el objeto proxy en la librería destino:

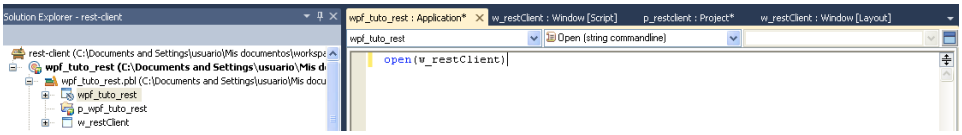


5. Dando forma a la ventana

Ahora que tenemos la aplicación cliente y el proxy generados, es momento de modificar la ventana donde aparecerán los datos. He optado por una solución donde se va a obtener los elementos descriptivos de la ruta: las indicaciones, y se mostrarán en un elemento dataWindow, cuando se pulse un botón. He aplicado también algo de estilo a la ventana. Queda de la siguiente forma:



Para poder ejecutar el proyecto, tendremos que modificar el evento open(string commandline) de la aplicaión. Para ello, haremos doble clic sobre la aplicación *wpf_tuto_rest*. Se nos abrirá por defecto el evento open(string commandline) asociado a la aplicación. Si aun así no aparece, lo seleccionaremos de la lista de eventos. Escribiremos lo siguiente en la ventana de script:



Esto abrirá nuestra ventana cuando se arranque la aplicación.

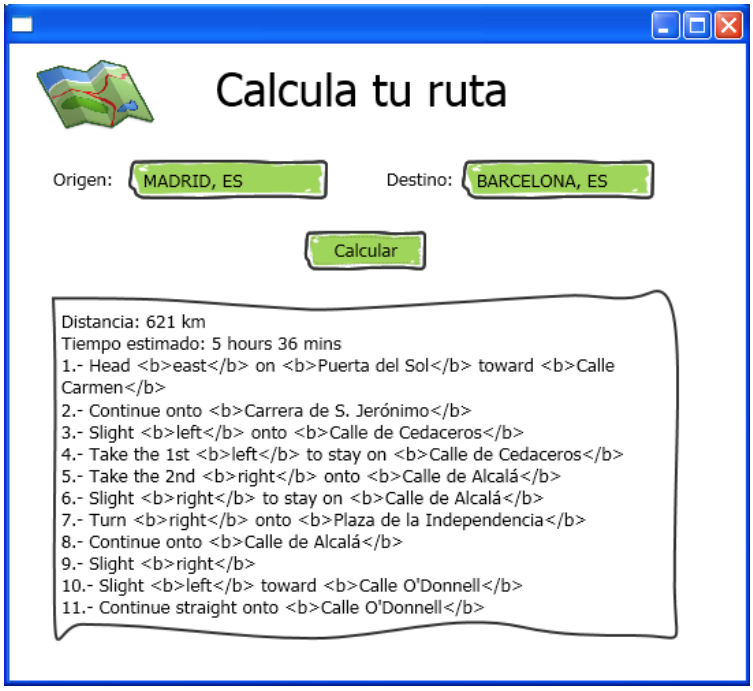
Nos queda lo más importante: Añadir la lógica de la aplicación a nuestro botón. Para ello, en la vista de diseño de la ventana haremos doble clic sobre el botón, y se nos abrirá la ventana de script con el evento Clicked() por defecto. El código del script será el siguiente:

```
view plain  print  ?
01.  //Variables locales
02.  google_rest_client_directions.root l_root
03.  google_rest_client_directions.rootItemItem legs[ ]
04.  google_rest_client_directions.rootItemItemItem steps[ ]
05.  integer i
06.  string rutas
07.
08.  DirectionsTutoService.DirectionsTutoProxy l_proxy
09.  l_proxy = create DirectionsTutoService.DirectionsTutoProxy
10.
11.  try
12.      l_root = l_proxy.GetMessage( )
13.
14.      legs = l_root.routes[1].legs
15.      steps = legs[1].steps
16.
17.      //Se asigna el valor del string con la lista de rutas
18.      if(steps.Length > 0) then
19.          rutas = "Distancia: " + legs[1].distance.text + "~n"
20.          rutas += "Tiempo estimado: " + legs[1].duration.text + "~n"
21.
22.          for i = 1 to steps.Length
23.              rutas += string(i) + ".- " + steps[i].html_instructions + "~n"
24.          next
25.
26.      else
27.          rutas = "No se ha obtenido ninguna ruta"
28.      end if
29.
30.      //Se establece el texto del control
31.      mle_rutas.Text = rutas
32.
33.  catch (System.Exception e)
34.      MessageBox("Error", "Error en la consulta de datos a Google")
35.  end try
```

6. Ejecutando la aplicación

Es hora de ejecutar nuestra aplicación y ver si los resultados son los correctos. Para ello hacemos clic en 'Debug' -> 'Run wpf_tuto_rest'.

Obtenemos el siguiente resultado en la consulta REST:



Posibles mejoras para la aplicación:

- Implementar la posibilidad de poder elegir otro destino (con argumentos en la llamada al servicio REST)
- Aplicar la región al servicio, ya que ahora está mostrándolo en el idioma por defecto (inglés)
- Eliminar los tags HTML, ya que Google representa así sus rutas (entiendo que es para mostrarlo adecuadamente en la web de Google Maps)
- Añadir la posibilidad de elegir el tipo de transporte usado. El API de Directions de Google tiene una lista de tipos de transporte. Podéis verlo aquí

Tenéis disponible la aplicación aquí

7. Conclusiones

PowerBuilder, aunque en principio nos puede dar la impresión de ser una herramienta "antigua", no se ha quedado ni mucho menos atrás, ya que nos ofrece muchas posibilidades a la hora de desarrollar aplicaciones propias o basadas en .NET, con soluciones optimizadas de conseguir la conectividad, mapeo y representación de datos a partir de bases de datos. Ofrece también bastantes soluciones de mercado (como por ejemplo la creación de servicios REST, pero también cualquier solución WCF o nativa), y proporciona una herramienta basada en Visual Studio muy intuitiva para el desarrollador, lo cual se agradece.

Además, PowerBuilder posee interfaces para la mayoría de bases de datos, además de soporte ODBC y OLE-DB. Tiene también bastantes herramientas, como Appeon Mobile, que corre aplicaciones PowerBuilder en iOS y Android. En resumen, una buena alternativa para el desarrollo de aplicaciones de negocio.

8. Referencias

- Web PowerBuilder SAP: http://www.sap.com/pc/tech/application-foundation-security/software/business-app-builder/index.html
- Powerscript reference: http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc37781.1250/html/psref/title.htm
- Links Yakov Werde: http://yakovwerde.ulitzer.com/node/1551687
- WPF: http://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx
- WCF: http://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx
- Directions API Google: https://developers.google.com/maps/documentation/directions/

A continuación puedes evaluarlo:

Regístrate para evaluarlo

Por favor, vota +1 o compártelo si te pareció interesante

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**