Guía de desarrollo, Anexo 03.08

Guía de programación en LoadRunner-Vugen 12.53

Autor: Oficina de Pruebas

GERENCIA INFORMÁTICA
JOSEFA VALCÁRCEL, 44
28027-MADRID

Índice General

1 INTRODUCCIÓN 5

1.1 OBJETIVO 5
1.2 AUDIENCIA 5
1.3 ESTRUCTURA DEL DOCUMENTO 5

2 HERRAMIENTA LOADRUNNER 6

2.1 ACCESO A LA APLICACIÓN 6
2.1.1 Acceso a la aplicación de Scripts (VuGen) 6
2.2 CONFIGURACIÓN GENERAL 7
2.2.1 Crear Scripts en ALM 7
2.2.2 Abrir Script Existente en ALM 12
2.2.3 Guardar Script Existente en ALM 17
2.3 REGLAS DE NOMENCLATURA 19
2.3.1 Reglas de Nomenclatura en scripts de carga (VuGen) 19
2.3.1.1 Transacciones (mediciones de tiempos de respuesta) 19
2.3.1.2 Parámetros de Entrada 21
2.3.1.3 Ficheros de Datos de Entrada 22
2.4 ESTRUCTURA DE DIRECTORIOS DE MONTAJE 24
2.4.1 Sintaxis 25
2.4.2 Ejemplo 26

3 NORMATIVA DE DESARROLLO DE SCRIPTS VUGEN 27

3.1 NORMATIVAS BÁSICAS 27
3.1.1 Protocolos 27
3.1.2 Think Times 28
3.1.3 Parametrización 29
3.1.4 Chequeos de Contenido en aplicaciones Web 32
3.1.5 Chequeos de contenido con WebServices 35
3.1.6 Mensajes 36
3.1.7 Resultados 36
3.1.8 Generation Log y Recording Log 38
3.2 NORMAS ADICIONALES 39
3.2.1 Protocolos: Web (HTTP/HTML) o Web Services 39
3.2.1.1 Caso: Chequeo en la descarga de un fichero 40
3.2.1.2 Caso: Cómo terminar la ejecución de un Usuario Virtual de forma ordenada 41
3.2.1.3 Caso: Uso de Librerías DLL externas 45
3.2.1.4 Caso: Crear Ficheros de Datos de Entrada dinámicamente, a través de la ejecución de Scripts SQL 46
3.2.1.5 Caso: Existe información dinámica que se genera dependiendo de los parámetros introducidos, para estos casos se utilizará la Correlación 49
3.2.1.6 Caso: Scripts que utilizan archivos locales en su grabación, para estos casos se utilizará Add Files. 51
3.2.1.7 Caso: Añadir Cabeceras HTTP en el Script 52
3.2.2 Protocolo: HTTP/HTML 54
3.2.2.1 Caso: Cómo grabar Scripts https con Fichero de Certificado Digital 54
3.2.2.2 Caso: Realizar búsquedas especiales en las llamadas a la Función web_reg_find() 57
3.2.2.3 Caso: Comprobar si una búsqueda ha tenido éxito 62
3.2.2.4 Caso: Ejecutar un Formulario Web con un Body dinámico 64
3.2.2.5 Caso: Realizar búsquedas condicionales en llamadas a la función web_reg_find() 67
3.2.2.6 Caso: Grabación de login mediante DataPower 68
3.2.2.7 Caso: Grabación de login mediante DataPower II 70
3.2.2.8 Caso: Grabación y ejecución de scripts con certificado digital sin WinInet 72
3.2.3 Protocolo: WebServices 73
3.2.3.1 Caso: Crear un Script sencillo de Protocolo WebServices con una llamada a la Función soap_request() 73
3.2.3.2 Caso: Ejecutar un WebService con un mensaje SOAP dinámico (parametrizar un XML) 78

3.2.3.3 Caso: Crear un Script sencillo de Protocolo WebServices securizado mediante certificado digital..... 86

Índice de Ilustraciones y Tablas

Ilustración 1. Acceso a Virtual User Generator 6
Ilustración 2. Creación de un nuevo script 7
Ilustración 3. Selección del protocolo 8
Ilustración 4. Acceso a la conexión de ALM 9
Ilustración 5. Ventana de conexión a ALM 9
Ilustración 6. Conexión a un proyecto de ALM 10
Ilustración 7. Opción de menú para salvar el script 11
Ilustración 8. Ventana para salvar el script 12
Ilustración 9. Ventana para conectar con ALM 13
Ilustración 10. Ventana para conexión con ALM 13
Ilustración 11. Ventana para seleccionar el proyecto de ALM 14
Ilustración 12. Ventana conexión establecida con proyecto ALM 15
Ilustración 13. Abrir script 16
Ilustración 14. Ventana selección scipt 16
Ilustración 15. Ventana de configuración 17
Ilustración 16. Compile 18
Ilustración 17. Opciones de menú 18
Ilustración 18. Listado de transacciones 19
Ilustración 19. Acceso secuencial 23
Ilustración 20. Acceso unico 24
Ilustración 21. Ventana protocolos I 27
Ilustración 22. Ventana protocolos II 28
Ilustración 23. Menú creación nuevo parámetro 30
Ilustración 24. Opciones de creación parámetro 30
Ilustración 25. Ventana de parámetros 31
Ilustración 26. Opciones de creación parámetro 31
Ilustración 27. Parámetro en el script 32
Ilustración 28. Página web 33
Ilustración 29. Opciones de configuración 37
Ilustración 30. Directorio de resultados 37
Ilustración 31. Ventana selección de protocolos I 39
Ilustración 32. Ventana selección de protocolos II 40
Ilustración 33. Diagrama de secuencia de generación datos 48
Ilustración 34. Opción de menú 50
Ilustración 35. Ventana de correlación 50
Ilustración 36. Valores identificados de correlación 51
Ilustración 37. Add extra files 52
Ilustración 38. Recording options. Port mapping 55
Ilustración 39. Ventana de Run-time settings 56
Ilustración 40. Opciones menú View 58
Ilustración 41. Step navigator 58

Ilustración 42. Snapshots 59
Ilustración 43. Detalle snapshot 60
Ilustración 44. Ventana opciones de chequeo 61
Ilustración 45. Ventana opciones correlación 62
Ilustración 46. Ventana configuración web_reg_find() 63
Ilustración 47. Ventana Run-Time Settings 64
Ilustración 48. Opción WinInet replay instead of Sockets (Windows only) 72
Ilustración 49. Ventana Create a New Script 75
Ilustración 50. Panel Solution Explorer 75
Ilustración 51. Opción para insertar un nuevo paso 76
Ilustración 52. Ventana Soap Request 77
Ilustración 53. Ejemplo petición soap_request 78
Ilustración 54. Ventana para salvar el script 81
Ilustración 55. Ventana para salvar el script 81
Ilustración 56. Ventana Manage Services 82
Ilustración 57. Ventana Import Service 83
Ilustración 58. Importación de un servicio 84
Ilustración 59. Opción SOA Tools 84
Ilustración 60. Ventana para seleccionar llamada a servicio 84
Ilustración 61. Editor xml 85
Ilustración 62. Contenido parametrizable 86

Tabla 1. Valores continuation_option de lr_exit 43
Tabla 2. Valores exit_status de lr_exit 43
Tabla 3. Ficheros para la generación dinámica de datos 49

1 INTRODUCCIÓN

1.1 Objetivo

El presente documento es la guía de programación de scripts de pruebas de carga en la herramienta **LoadRunner 12.53.**

Esta guía **no es un Manual de Programación** de LoadRunner. Los manuales oficiales de LoadRunner los podemos encontrar en la ayuda on-line de la herramienta.

Las reglas de nomenclatura, recomendaciones y procedimientos que aquí se exponen son de obligado cumplimiento para todas aquellas personas que vayan a desarrollar scripts en LoadRunner VuGen.

1.2 Audiencia

Este documento está dirigido a todas las personas vinculadas o que colaboren en labores relacionadas con la gestión, análisis, diseño, desarrollo, auditoría, implantación y explotación de los sistemas de información de la gerencia de informática de la Dirección General de Tráfico.

1.3 Estructura del documento

Este documento está distribuido en 3 capítulos, con los siguientes contenidos:

- Capítulo 1: Introducción, contiene información relativa al propio documento.
- Capítulo 2: Herramienta, contiene información general del uso de la herramienta Vugen.
- Capítulo 3: Normativa, contiene información de la normativa de uso de la herramienta Vugen para las aplicaciones DGT.

2 HERRAMIENTA LOADRUNNER

2.1 Acceso a la aplicación

2.1.1 Acceso a la aplicación de Scripts (VuGen)

El acceso a la aplicación se realizará haciendo doble clic sobre el icono del escritorio o mediante "**Inicio\Todos Los Programas\ HP Software\HP LoadRunner\Virtual User Generator**".

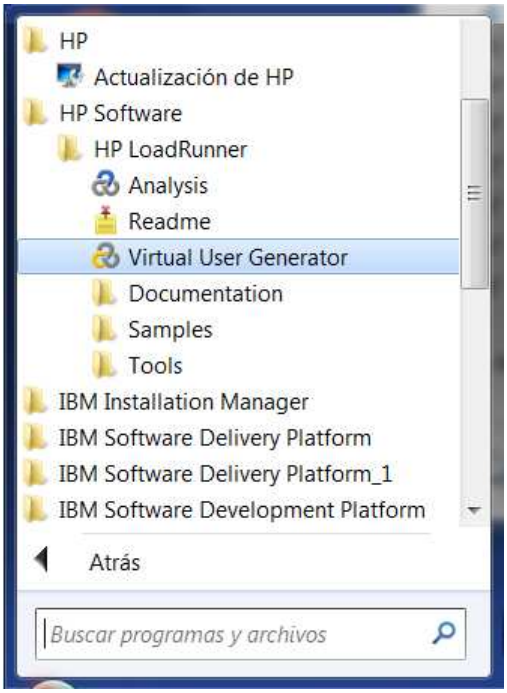


Ilustración 1. Acceso a Virtual User Generator

2.2 Configuración General

2.2.1 Crear Scripts en ALM

Los pasos para crear un nuevo script, NO existente en ALM son:

Ir a menú y ejecutar “File\New Script and Solution”.

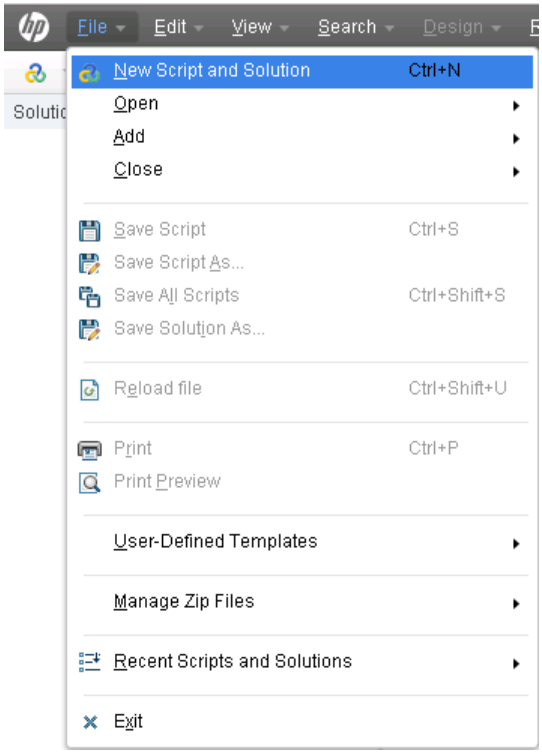


Ilustración 2. Creación de un nuevo script

Seleccionar el tipo de protocolo. Por ejemplo: “**Web - HTTP/HTML**”.

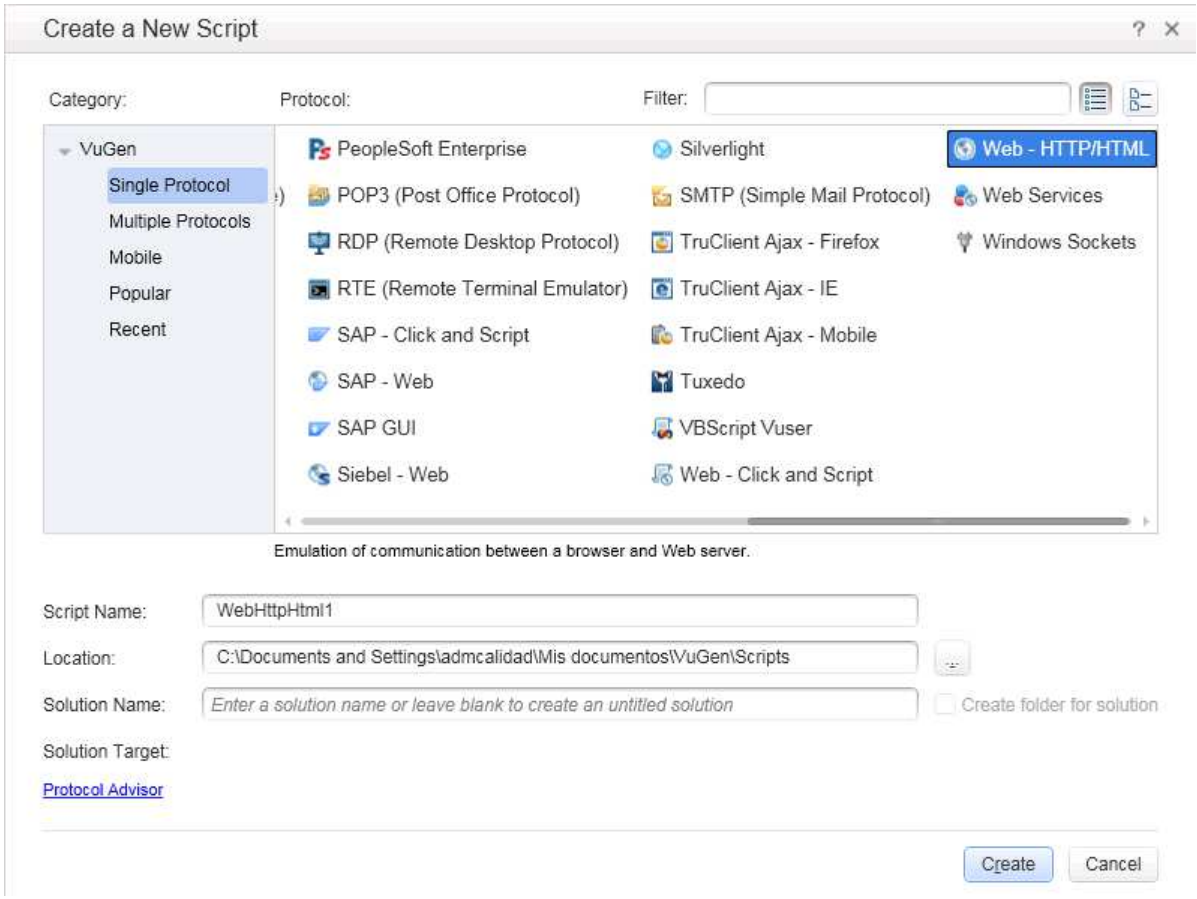


Ilustración 3. Selección del protocolo

Pulsar en el botón “**Create**”.

Grabar el script siguiendo las indicaciones dadas en esta guía.

Una vez finalizada la grabación se deben realizar los siguientes pasos para salvar el script:

Conectarse a ALM pulsando en el menú “**Version Control\ALM\ALM Connection…**”.

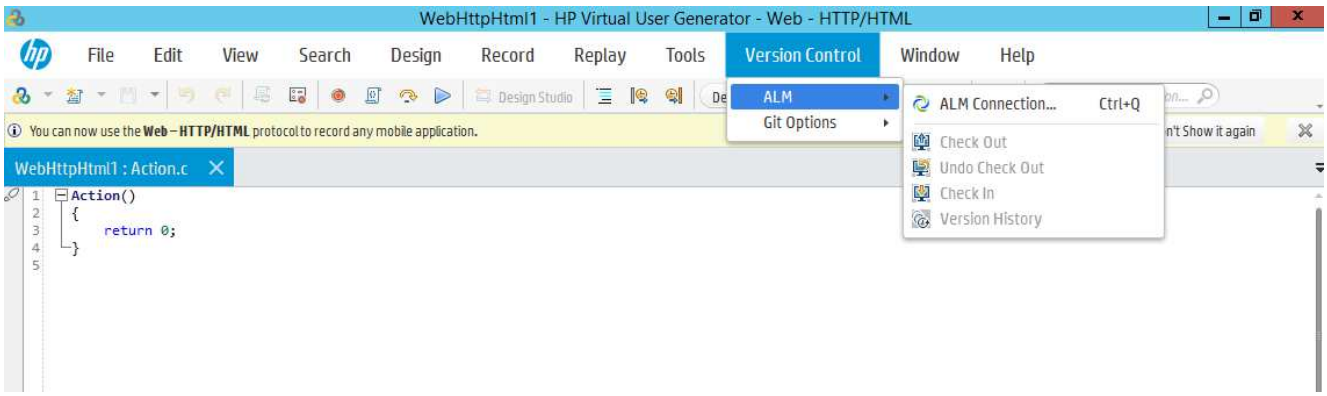


Ilustración 4. Acceso a la conexión de ALM

En la siguiente pantalla, introducir la ruta del portal de ALM en el campo "**Server URL**".

Introducir el nombre de usuario y password en los campos "**User name**" y "**Password**".

Pulsar en el botón "**Connect**".

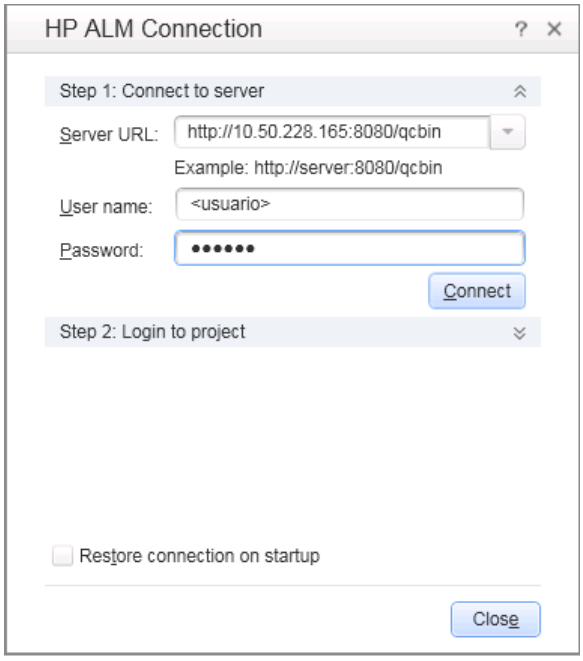


Ilustración 5. Ventana de conexión a ALM

En el desplegable “**Project**” seleccionar el nombre del proyecto al cual nos queremos conectar.

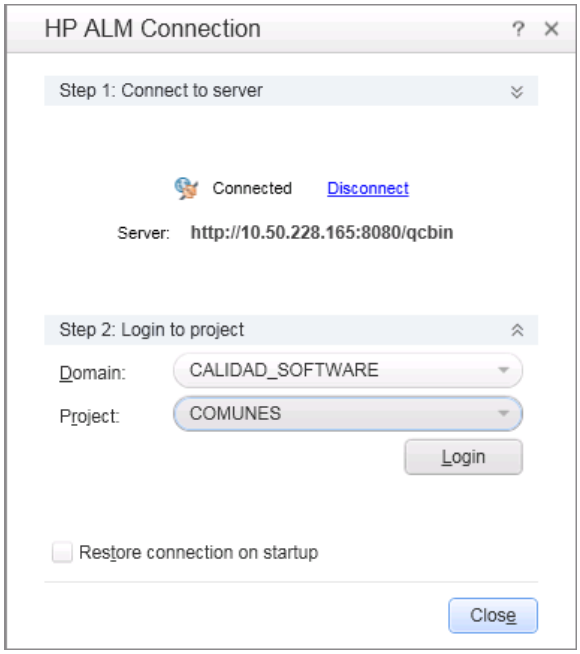


Ilustración 6. Conexión a un proyecto de ALM

Pulsar en el botón “**Login**”.

Una vez conectados, pulsar en el botón “**Close**”.

Ir al menú y ejecutar “**File\Save Script As …**”.

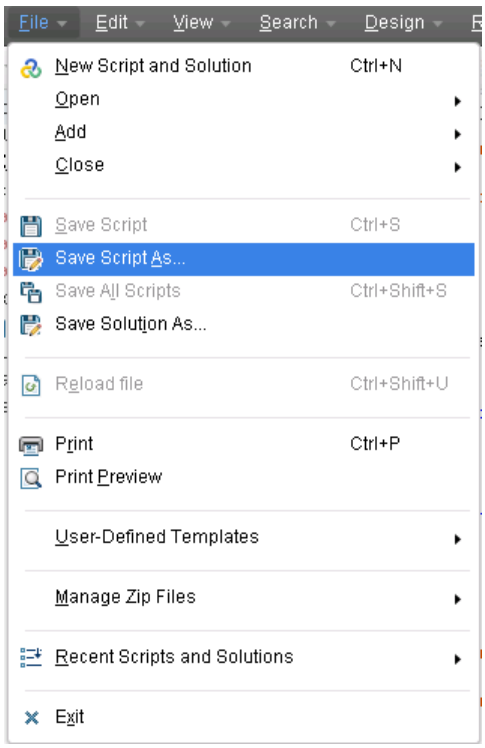


Ilustración 7. Opción de menú para salvar el script

Hacer click en "**ALM Test Plan**" y navegar por la estructura hasta localizar la ubicación donde se va a guardar el script.

En el campo "**File Name**" introducir el nombre del script siguiendo la nomenclatura especificada en este documento.

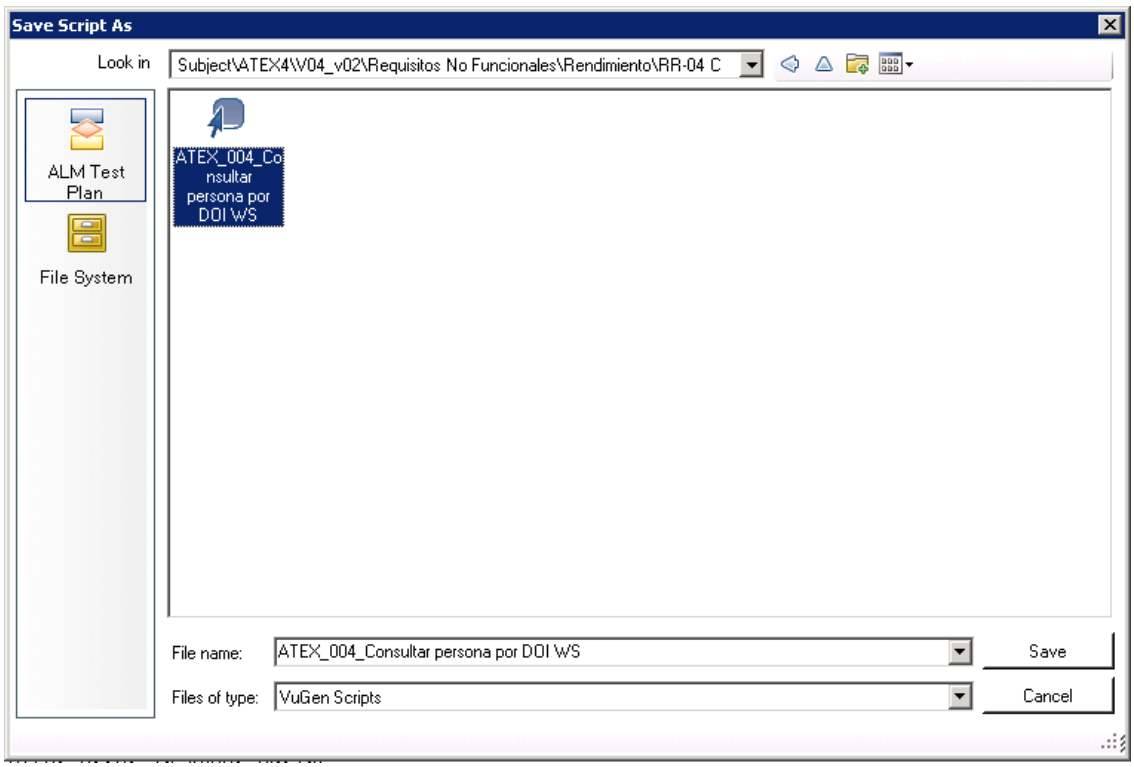


Ilustración 8. Ventana para salvar el script

Pulsar en el botón "**Save**".

2.2.2 Abrir Script Existente en ALM

Para abrir los scripts de carga ubicados en el portal ALM los pasos a realizar son:

Conectarse a ALM pulsando en el menú "**Version Control\ALM\ALM Connection…**".

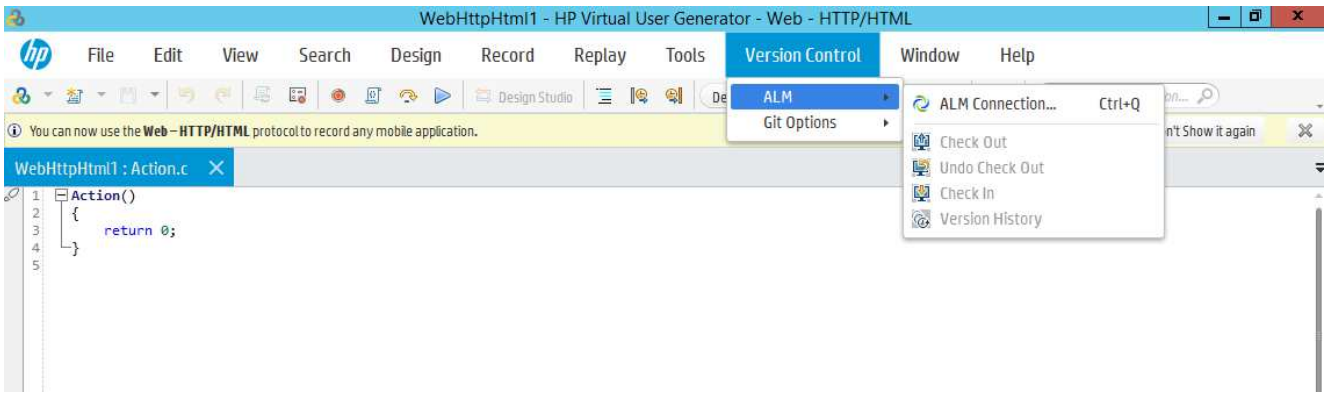


Ilustración 9. Ventana para conectar con ALM

En la siguiente pantalla, introducir la ruta del portal de ALM en el campo "**Server URL**".

Introducir el nombre de usuario y password en los campos "**User name**" y "**Password**".

Pulsar en el botón "**Connect**".

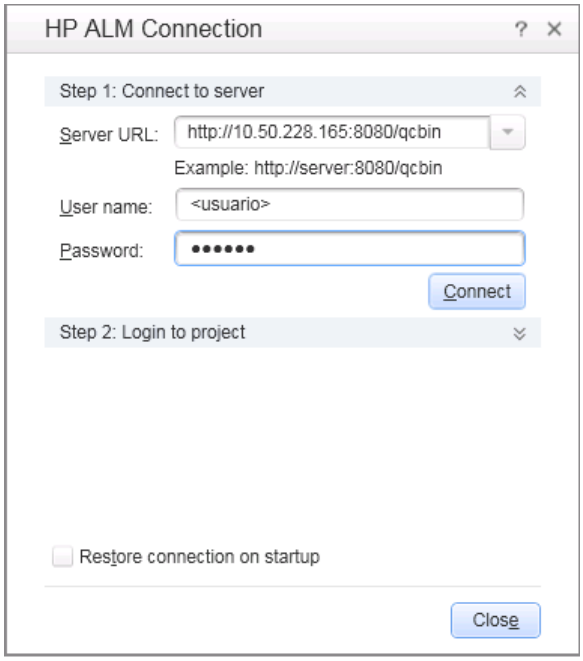


Ilustración 10. Ventana para conexión con ALM

En el desplegable "**Project**" seleccionar el nombre del proyecto al cual nos queremos conectar.

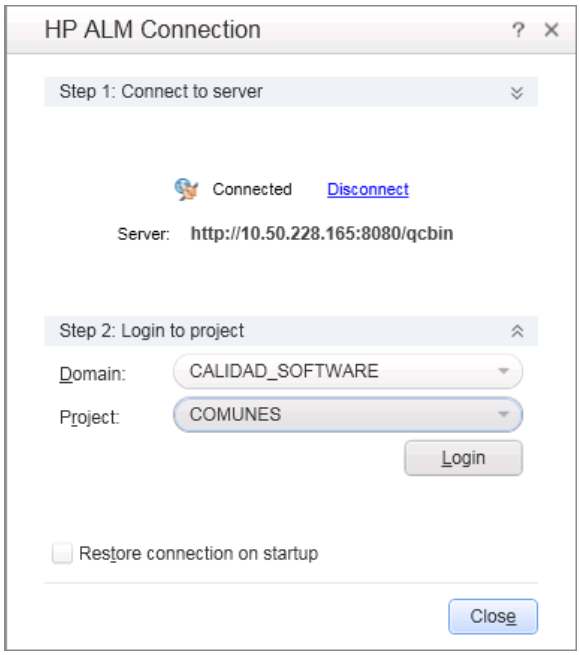


Ilustración 11. Ventana para seleccionar el proyecto de ALM

Pulsar en el botón "**Login**".

Una vez conectados, pulsar en el botón "**Close**".

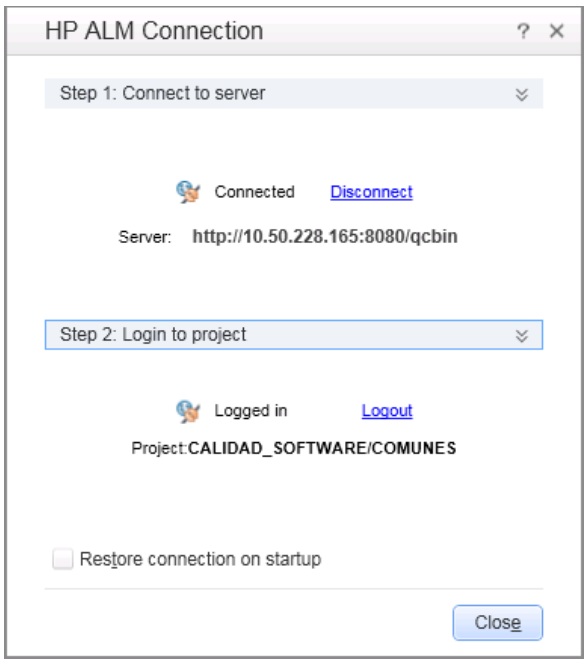


Ilustración 12. Ventana conexión establecida con proyecto ALM

Ir al menú a la opción "**File\Open\Script/Solution**".

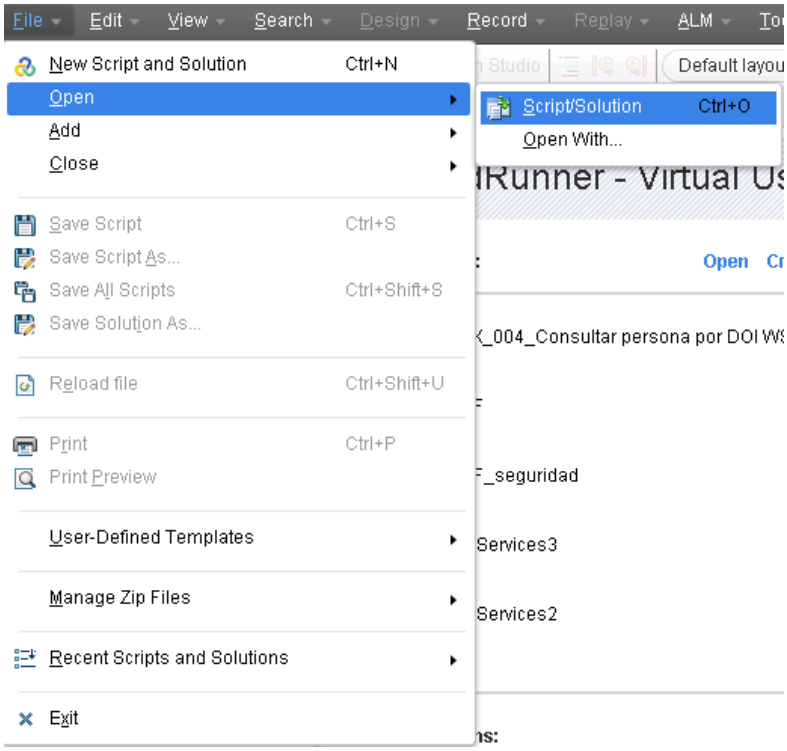


Ilustración 13. Abrir script

Dentro de "**ALM Test Plan**" navegar por la estructura de carpetas hasta seleccionar el script a abrir.

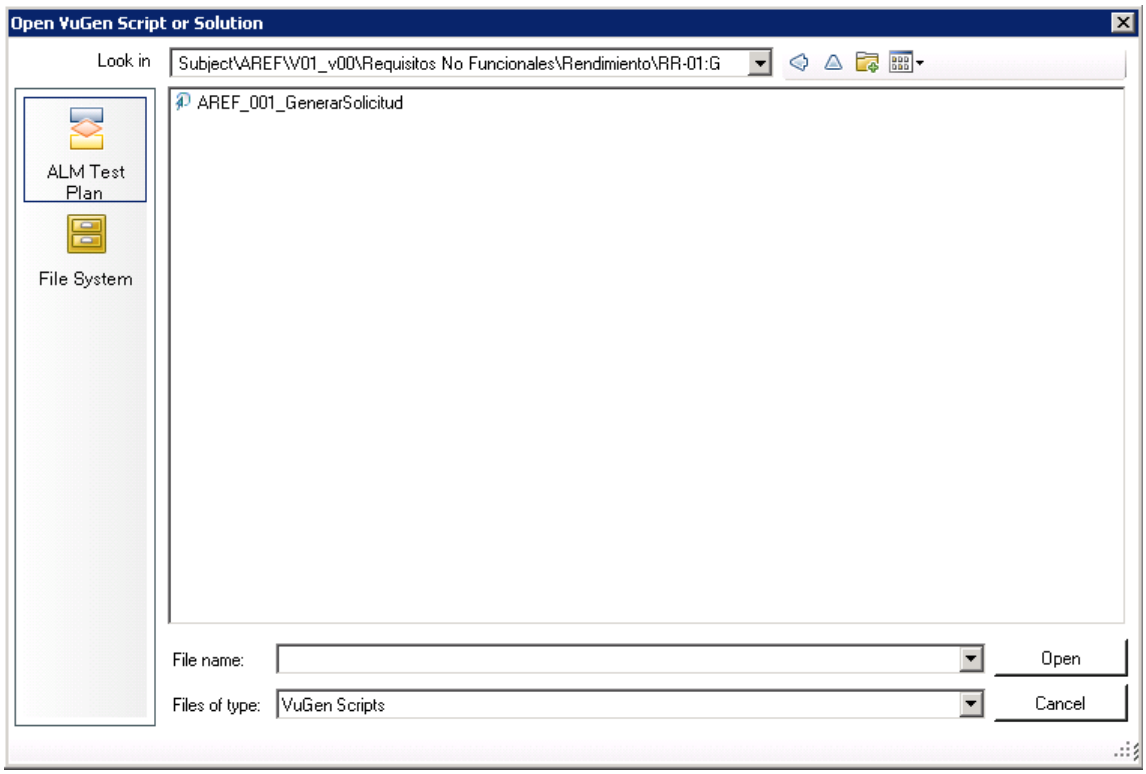


Ilustración 14. Ventana selección scipt

Pulsar en el botón "**Open**".

Grabar el script de carga siguiendo la nomenclatura y especificaciones dadas en ese documento.

Al abrir un script de ALM, asegurarse que sólo se van a grabar los resultados de la última de las ejecuciones. Para ello se debe asegurar que, antes de ejecutar el script, en las opciones no está seleccionada '**Enable results of replay summary to be saved to a named folder after each script run'**. Para ello:

Ir a "**Tools/Options**", sección "**Scripting"** en la pestaña "**Replay**".

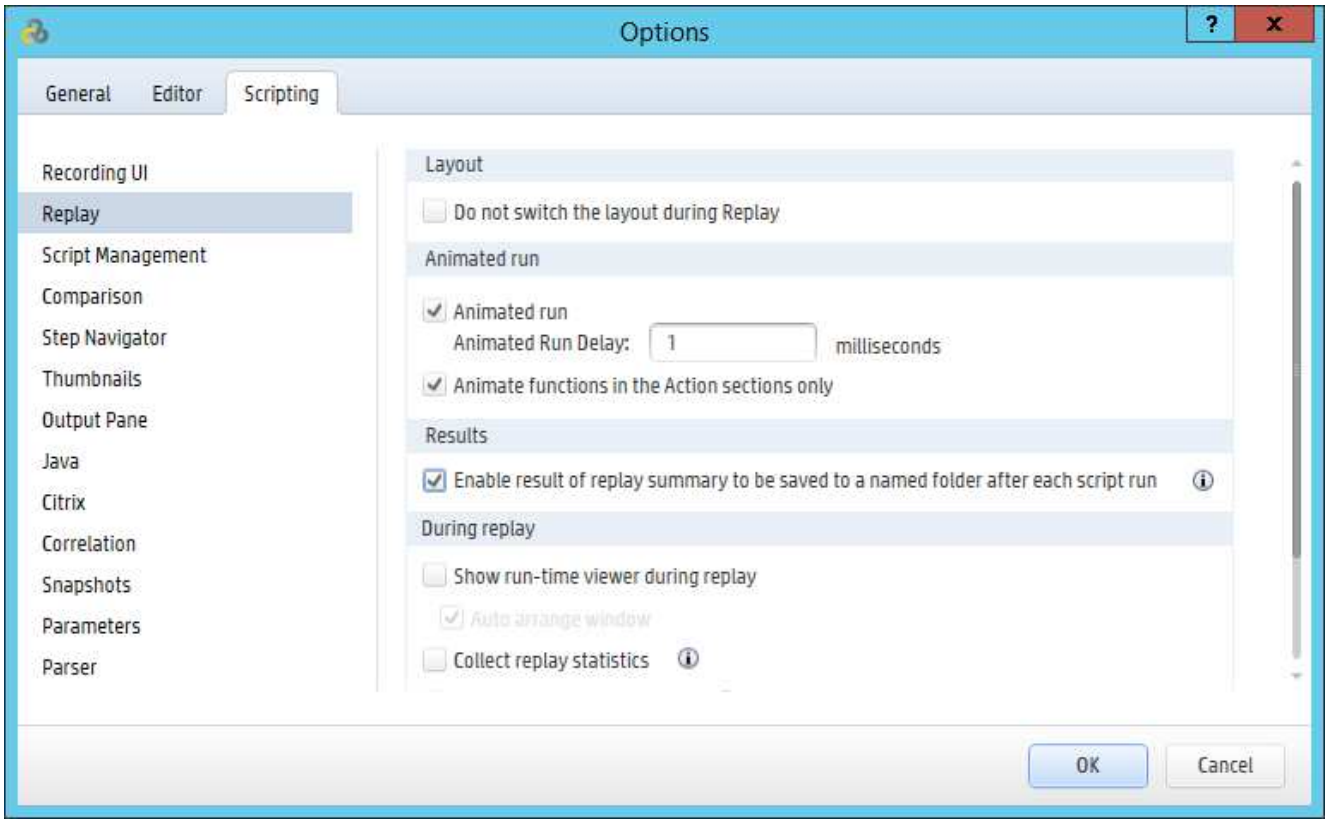


Ilustración 15. Ventana de configuración

Una vez se ejecute el script, los resultados de una ejecución sobrescribirán los de la ejecución anterior y en el momento de subirlo a ALM sólo se grabará la última de las ejecuciones.

No se podrá iterar más de 10 veces un script y nunca con el log extendido, para que los resultados que se suben a ALM no sean muy pesados. En el caso de que se necesite iterar más veces un script, se debe guardar en local y ejecutarlo las veces que sea necesario.

2.2.3 Guardar Script Existente en ALM

Después de abrir el script creado desde ALM, se procede a grabar la secuencia de acciones a ejecutar. Cuando finaliza la grabación y al ser un script ya existente en ALM deberemos realizar los siguientes pasos:

Ir al menú "**Replay\Compile**".

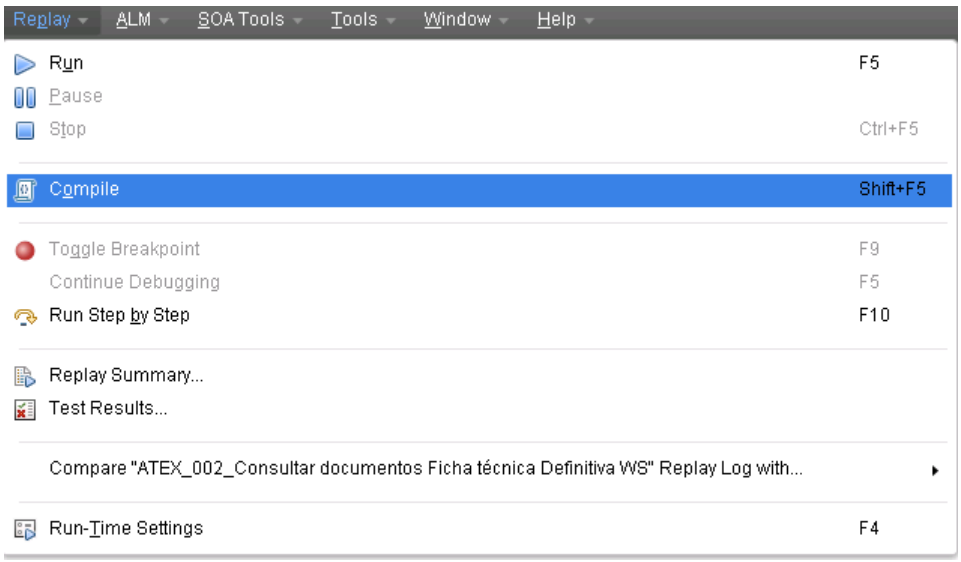


Ilustración 16. Compile

Ir al menú "**File\Save Script**" o pulsar en el botón de la barra de herramientas.

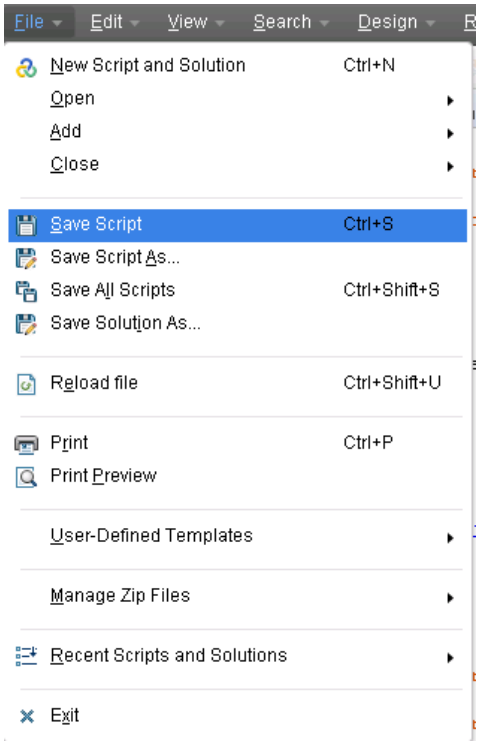


Ilustración 17. Opciones de menú

2.3 Reglas de Nomenclatura

2.3.1 Reglas de Nomenclatura en scripts de carga (VuGen)

2.3.1.1 Transacciones (mediciones de tiempos de respuesta)

En LoadRunner las **transacciones** son los segmentos de código de un script en los que se quieren realizar mediciones de Tiempos de Respuesta. Es importante **no confundir estas transacciones con los scripts de LoadRunner**, es decir, con las transacciones de negocio.

Por defecto, en **LoadRunner** se definen estas tres transacciones:

Vuser_init_Transaction **.** Acciones que se realizan al principio de la ejecución del script, como por ejemplo, el Login.

Action_Transaction. El resto de Acciones que se realizan en la ejecución del script

Vuser_end_Transaction. Acciones que se realizan al final de la ejecución del script, como por ejemplo, desconectarse de la Aplicación.

Nosotros podemos definir nuestras propias transacciones para medir el Tiempo de Respuesta de una parte de la ejecución del script, por ejemplo, el tiempo que tarda en generarse un listado.

En la siguiente imagen podemos apreciar los resultados de la ejecución de un script en el que se han definido varias transacciones. Las transacciones señaladas con un rectángulo en **rojo** son las tres transacciones por defecto que crea LoadRunner. El resto son transacciones definidas por el usuario:

Transaction Name	SLA Status	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop
Action_Transaction	-	1.81	4.743	31.503	5.268	4.591	106	0	20
TRAN_DWEB_000_Pagina_Inicio	-	0.144	0.26	0.638	0.066	0.301	73	0	0
TRAN_GEUS_001_Buscar_Usuario	-	0.154	0.199	0.354	0.04	0.259	28	0	0
TRAN_WREL-CAU_000_IniciarSesion	-	0.157	0.234	0.478	0.081	0.33	25	0	0
TRAN_WREL-CAU_001_ConsultaRegistros	-	0.381	0.5	0.757	0.101	0.642	19	0	1
TRAN_WREL-CAU_001_DescargarDocumento	-	0.229	0.293	0.396	0.064	0.396	5	0	0
TRAN_WREL-CAU_001_DetalleRegistro	-	0.651	0.789	1.242	0.164	0.953	12	0	0
vuser_end_Transaction	-	0	0	0	0	0	24	0	0
vuser_init_Transaction	-	0	0.012	0.27	0.054	0.004	24	0	0

Ilustración 18. Listado de transacciones

2.3.1.1.1 Tipos de Transacciones: A nivel de Script aislado y a nivel de Escenario

Consideramos que una misma Transacción puede tener sentido no sólo **a nivel de un script aislado**, sino también **a nivel de escenario dentro de una misma aplicación**.

Es decir, si por ejemplo, en un mismo Escenario de la Herramienta Controller lanzamos más de un script de una misma Aplicación, y en todos estos scripts se realiza el Login, entonces podemos dar el mismo nombre a la Transacción de Login (por ejemplo TRAN_APLI_000_Pagina_Inicio) dentro de cada uno de los scripts. De esta manera, podemos obtener el Tiempo de Respuesta de la ejecución del Login a nivel de Escenario, y en consecuencia, de la Aplicación. Esto es una gran ventaja desde el punto de vista funcional ya que podemos saber el Tiempo de Respuesta de una parte de la Aplicación, de una funcionalidad, ejecutando diferentes transacciones de negocio.

Asimismo, utilizando esta técnica también podremos saber el Tiempo de Respuesta a nivel de script aislado, a nivel de grupo de scripts, a nivel de máquina en la que se ejecuta el script, a nivel de usuario virtual, etc… ya que la herramienta Analysis de LoadRunner nos permite desglosar los resultados hasta estos niveles.

Debemos asegurarnos que **nunca** haya una sentencia **lr_think_time** entre la apertura y el cierre de una transacción, ya que sino el tiempo medido por dicha transacción quedará desvirtuado.

2.3.1.1.2 Sintaxis

TRAN_<Aplicación_Subsistema>_999_<Descripción>

Donde:

Aplicación	**Es el acrónimo que asigna la DGT a cada Aplicación**	
Subsistema	**Es el acrónimo del subsistema**	
999	**Es el índice de la Transacción, cuyo valor puede ser uno de los siguientes:**	
	000	**Si queremos medir el Tiempo de Respuesta a nivel de Escenario, de Aplicación**
	El índice del Script de LoadRunner	**Si queremos medir el Tiempo de Respuesta a nivel de Script aislado**
Descripción	**Es la descripción de la Transacción**	
	<NOTA>: No utilizar acentos ni caracteres extraños	

2.3.1.1.3 Ejemplos

Ejemplo 1:

TRAN_APLI_000_Pagina_Inicio → Script APLI_001_ConsultaWeb

TRAN_APLI_000_Pagina_Inicio → Script APLI_002_ModificacionWeb

En este ejemplo queremos medir el Tiempo de Respuesta a nivel de Escenario, de Aplicación. Es decir, varios scripts (varias transacciones de negocio) dentro del Escenario ejecutarán la Página de Inicio (TRAN_APLI_000_Pagina_Inicio), es decir, la misma funcionalidad.

Los Tiempos de Respuesta obtenidos se acumulan bajo una misma medición (*Measurement*) cuyo nombre es: TRAN_APLI_000_Pagina_Inicio, aunque podemos desglosarlos a nivel de script aislado, a nivel de grupo de scripts, a nivel de máquina en la que se ejecuta el script, a nivel de usuario virtual, etc…

Ejemplo 2:

TRAN_APLI_001_BuscarUsuario –> Script APLI_001_BuscarUsuario

En este otro ejemplo queremos medir el tiempo de respuesta a nivel de script aislado. Un sólo script ejecutará la transacción. En concreto el script de LoadRunner cuyo índice es el **00X** dentro de la aplicación**.**

NOTA: Todas las peticiones que se realicen al servidor deben ir englobadas dentro de una TRAN_APLI_SUBS_00X_XXXX

2.3.1.2 Parámetros de Entrada

2.3.1.2.1 Sintaxis

dp<Nombre del campo>	**Se corresponde con el identificador de campo del fichero de datos de entrada. Es un campo único a nivel de script**

2.3.1.2.2 Ejemplos

dpURL

dpPassword

2.3.1.3 Ficheros de Datos de Entrada

Ver documento: Guía Datos de Entrada Pruebas.doc

2.3.1.3.1 Modos de acceso al fichero

Una vez creado el fichero de datos de entrada con los parámetros de entrada es necesario configurar el modo de acceso a los mismos para la ejecución de la prueba. Es necesario que la configuración de acceso a los datos que se utilice sea la mas adecuada para simular una situación lo mas realista posible, y de acuerdo a la funcionalidad que ejecute el proceso de negocio.

Existen cuatro modos de acceso posibles:

Sequential. Registro a registro desde el principio del fichero.

Random. Registros sin seguir un orden.

Unique. Registros que no se repiten.

Same line as <parameter>. Mismo acceso que otro parámetro.

A continuación se muestra un ejemplo de cómo se realiza el acceso a los datos del fichero en una prueba de carga según el modo seleccionado.

Sequential

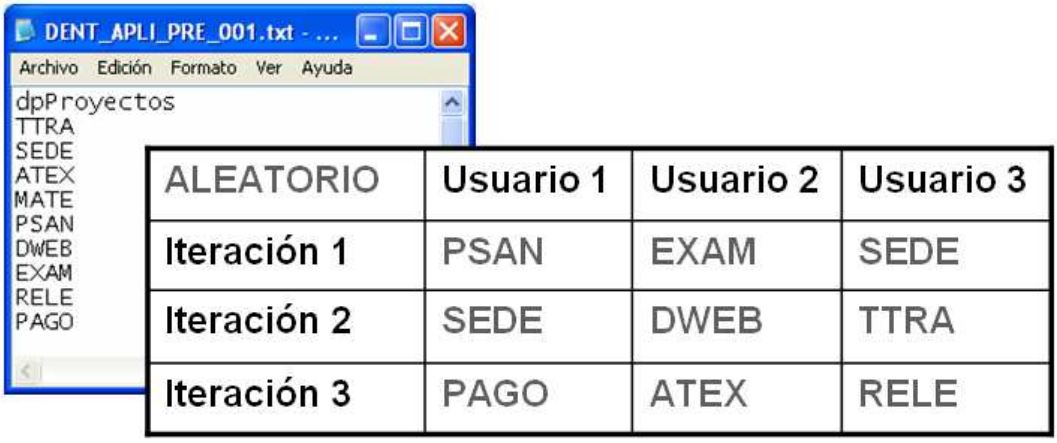
En el método “Sequential” cada usuario virtual utiliza una de las filas del fichero de datos en cada iteración empezando por la primera hasta la última.

Ilustración 19. Acceso secuencial

Este método se utiliza cuando se tiene una fila de datos para cada iteración de los usuarios virtuales, sin existir la necesidad de datos únicos, y siempre que no tenga efecto la caché de datos.

Random

En el método “Random” cada usuario virtual utiliza una fila aleatoriamente en cada iteración de la prueba. En este modo de acceso no se garantiza la unicidad de los datos y puede ser que dos usuarios utilicen la misma fila de datos en el mismo momento.



Este método se utiliza cuando se quiere simular la aleatoriedad de datos sin tener la restricción de datos únicos. Es un método muy aconsejado en los procesos de negocio de consulta.

Unique

En el método “Unique” el fichero de datos es dividido en secciones basándose en el número de usuarios de la prueba. Cada usuario utiliza una sección diferente del fichero y una fila de datos distinta en cada iteración, por lo que dos usuarios nunca utilizan la misma fila de datos durante la prueba.

Ilustración 20. Acceso unico

Este método se utiliza cuando existe una restricción de unicidad en los datos del proceso de negocio. Los datos en si mismos no son chequeados, es necesario asegurarse que no existe duplicidad de datos en el fichero.

Same line as <parameter>

En el método “Same line as <parameter>” se asigna el dato desde la misma fila del cual lo toma un parámetro previamente definido. Este parámetro sería el campo clave del fichero y debe tener asignado al menos uno de los tres métodos anteriores, Sequential, Random o Unique.

Este método se utiliza cuando existe una dependencia entre los datos, como es el caso del usuario y password.

2.4 Estructura de Directorios de montaje

Las pruebas de carga de cada aplicación están formadas por dos tipos de Componentes:

Componentes nativos de LoadRunner VuGen (Scripts, Ficheros de Datos de Entrada, Escenarios…). Estos componentes se almacenan y gestionan en ALM. Se recomienda ver los documentos asociados a la gestión y uso de ALM.

Componentes externos. Por ejemplo, Scripts SQL para la generación dinámica de ficheros de datos de entrada, Certificado que hay que instalar en el navegador,… Estos componentes se almacenan y gestionan en ALM. Se recomienda ver los documentos asociados a la gestión y uso de ALM.

Cuando el volumen de datos de entrada es elevado (superior a 8 Mb) estos componentes se almacenan en un ZIP en ALM a nivel de caso de prueba y se gestionan en el disco en local, organizados por aplicación y versión.

La estructura de Directorios de montaje donde se deben ubicar este tipo de caso se detalla a continuación.

2.4.1 Sintaxis

C:\<Aplicación>_<Subsistema> (Directorio raíz. Montarlo siempre en C:\)

+---<Aplicación>_<Subsistema>-.x.y (Versión de la Aplicación)

+--------LR

+--------data (Ficheros de Datos de Entrada)

Donde:

Aplicación	**Es el acrónimo que asigna la DGT a cada Aplicación**
Subsistema	**Es el acrónimo que se asigna a cada subsistema de la aplicación.**
x.v	**Es la Versión**

2.4.2 Ejemplo

<NOTA>: La Aplicación es **CCAL**

C:_CCAL\CCAL-1.0\LR\data\

xml_solicitud_00001.xml

xml_solicitud_00002.xml

…

xml_solicitud_99998.xml

xml_solicitud_99999.xml

3 NORMATIVA DE DESARROLLO DE SCRIPTS VUGEN

3.1 Normativas básicas

3.1.1 Protocolos

Los únicos protocolos con los que se permite grabar los scripts de carga son **Web (HTTP/HTML)** o **WebService.**

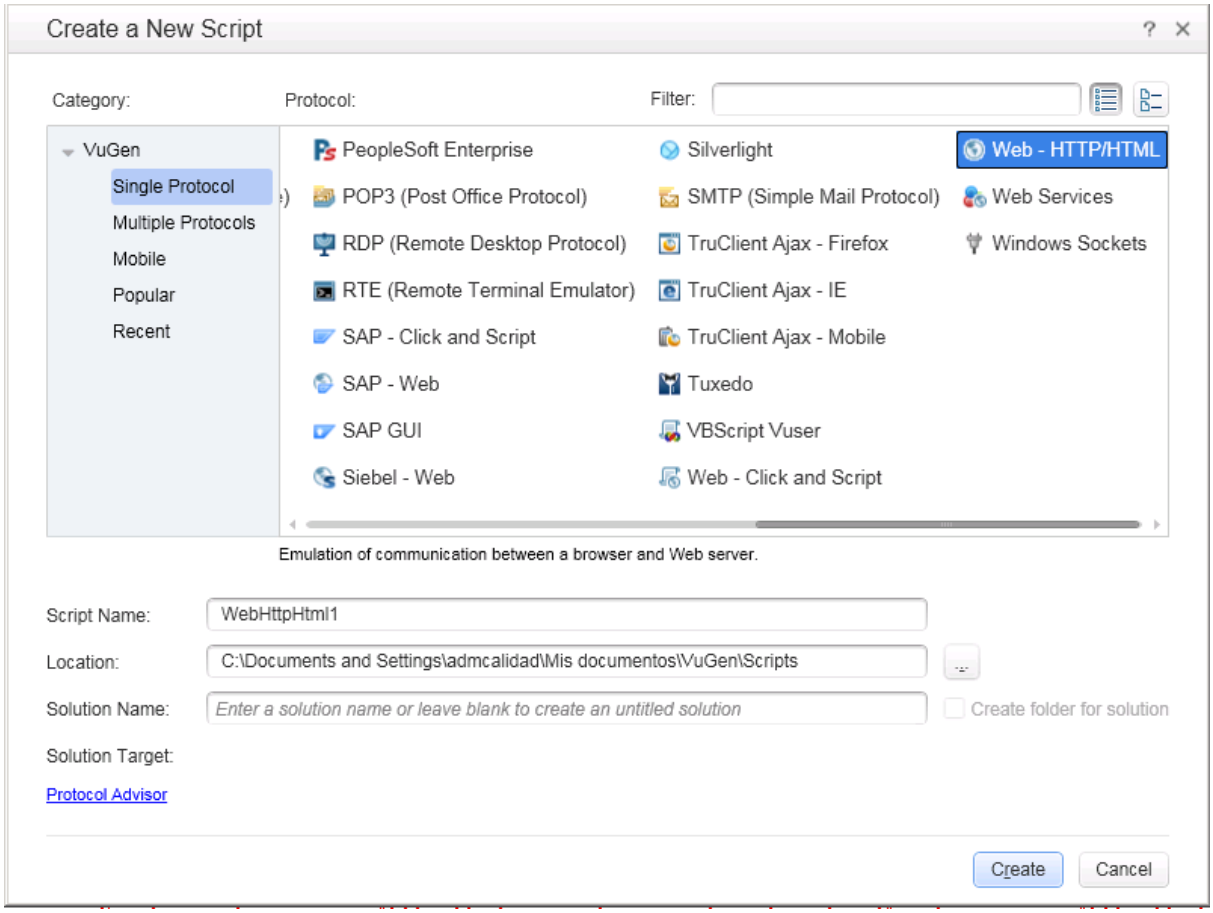


Ilustración 21. Ventana protocolos I

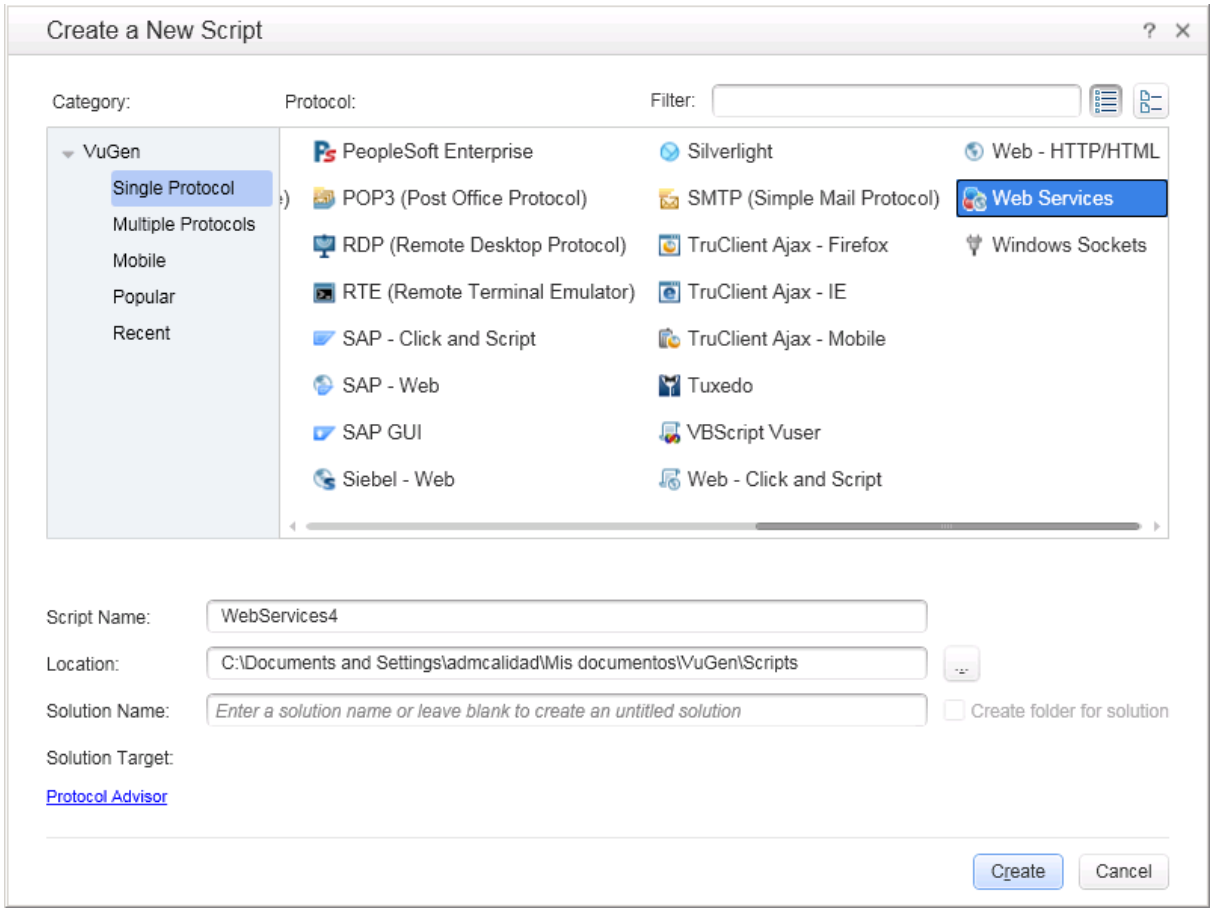


Ilustración 22. Ventana protocolos II

3.1.2 Think Times

Los **Think Times** son los tiempos de espera de Usuario. Por ejemplo, el tiempo que transcurre mientras un Usuario rellena un Formulario Web y hasta que pulsa el botón "**Aceptar**".

Los Think Time que se incluyan en los scripts deben ser reales para que la ejecución del mismo sea lo mas realista posible y **nunca** deben encontrarse dentro de una transacción.

Mediante la sentencia **lr_think_time(t)** se incluyen los tiempos de espera en el script, siendo 't' el tiempo en segundos que esperará hasta que realice la siguiente acción.

Si no ponemos *Think Times* en los scripts, los resultados obtenidos no se ajustarán a la realidad, dado que no estaremos simulando la actividad real de los Usuarios.

NOTA: Es obligatorio utilizar los Think Times que se producen durante la grabación del script.

3.1.3 Parametrización

En la grabación de una transacción de negocio con la herramienta VuGen, se capturan valores constantes como por ejemplo, el login a una aplicación, en el que necesitamos especificar el *usuario* y el *password* para poder acceder. Además, necesitaremos que la transacción de negocio se ejecute con diferentes usuarios. Para lo cual necesitamos parametrizar nuestro script reemplazando dichos valores por parámetros de LoadRunner.

Al parametrizar los datos que utilizamos en el script evitamos que haya cacheo de los datos en los servidores, con lo que la prueba es mucho más realista.

NOTA: Debe parametrizarse todos los datos susceptibles de cambio para asegurar la misma variabilidad de datos que tendrá el proceso de negocio en producción.

El proceso a seguir es el siguiente:

En el script de LoadRunner que se genera, se selecciona la cadena de texto que queremos parametrizar, pulsamos botón derecho y seleccionamos la opción "***Replace with parameter***".

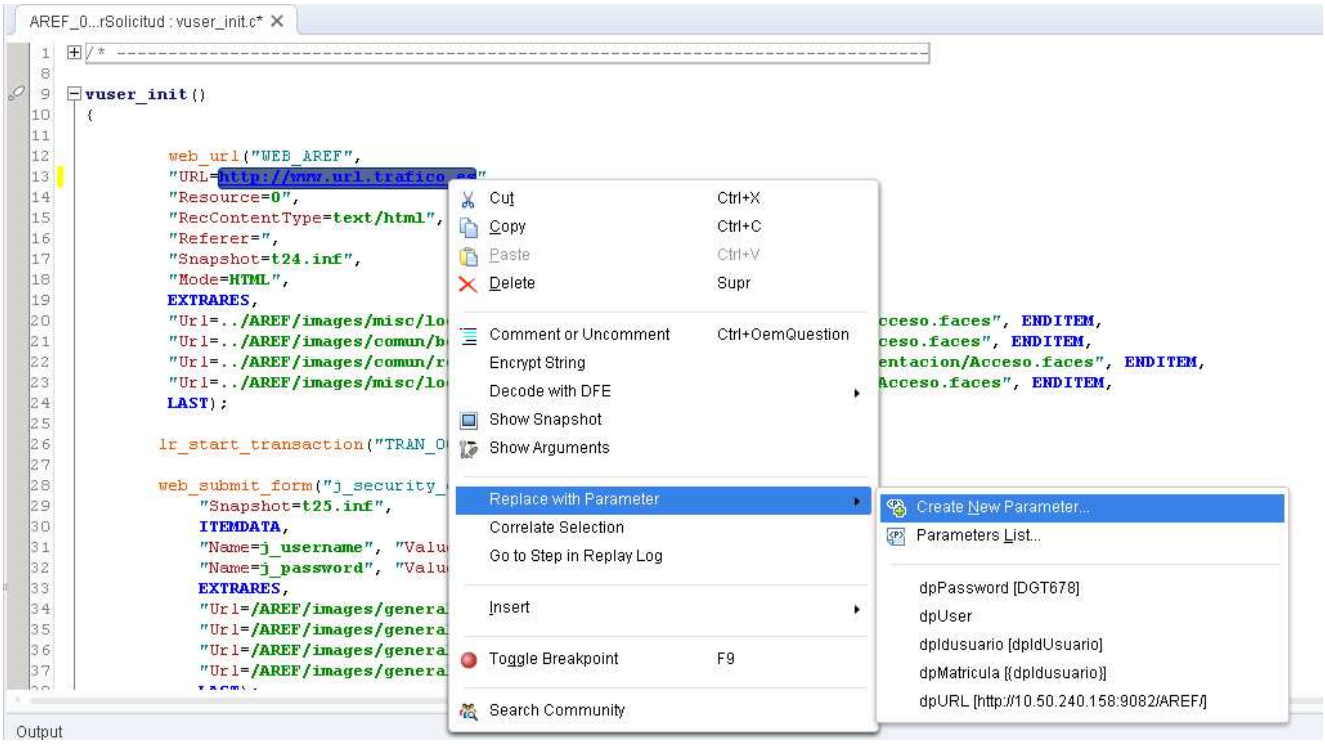


Ilustración 23. Menú creación nuevo parámetro

Nos aparecerá la siguiente pantalla:

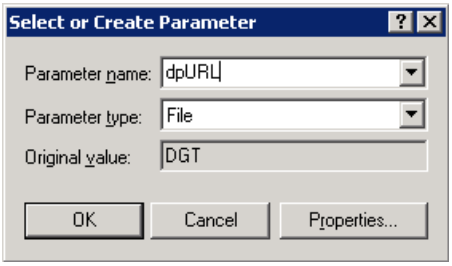


Ilustración 24. Opciones de creación parámetro

En el campo "**Parameter name**" introducimos el prefijo "**dp**" seguido del nombre del parámetro y pulsamos en "**Properties**". Ver apartado Nomenclatura Parámetros de Entrada.

En el campo "**File path**" introducimos el nombre del fichero que contienen los parámetros de Entrada.

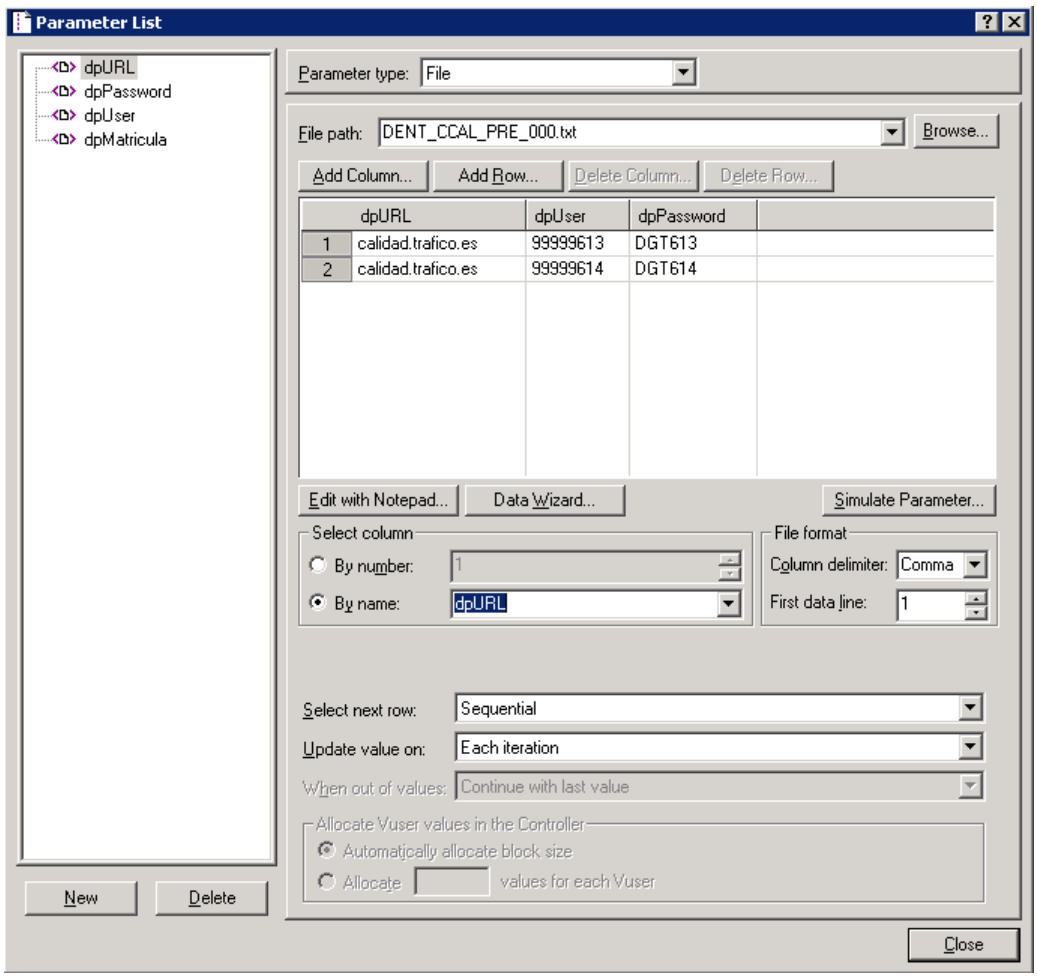


Ilustración 25. Ventana de parámetros

Pulsamos sobre el botón "**Close**" y después "**OK**".

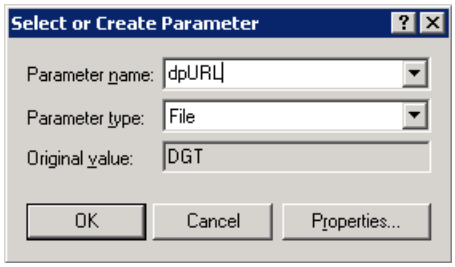


Ilustración 26. Opciones de creación parámetro

En el código del script generado en VuGen, nos aparecerá el nombre del parámetro por el que lo hemos sustituido.

```
AREF_0...rSolicitud : vuser_init.c
1   /* ----------------------------------
8
9   vuser_init()
10  {
11
12          web_url("WEB_AREF",
13          "URL={dpURL}", |
14          "Resource=0",
15          "RecContentType=text/html",
16          "Referer=",
17          "Snapshot=t24.inf",
18          "Mode=HTML",
```

Ilustración 27. Parámetro en el script

Después de añadir el parámetro se deberá verificar que el script funciona correctamente.

3.1.4 Chequeos de Contenido en aplicaciones Web

Un chequeo de contenido es buscar un texto o una imagen (dentro de una página HTML, etc…) que certifique que la transacción de negocio se ha ejecutado correctamente.

Por ejemplo, realizamos la petición de la página Web de Incidencias de circulación a la aplicación de la Web de la DGT en Internet. El servidor nos devuelve la siguiente página:

Ilustración 28. Página web

Para verificar que la Transacción de negocio se ha realizado correctamente, es decir, que el Servidor nos ha devuelto la página correcta, tenemos que buscar un texto, imagen,… que verifique que el proceso se ha ejecutado correctamente. Por ejemplo el siguiente *String* "*Últimas incidencias de circulación*"

Si se encuentra el *contenido a chequear*, la ejecución de la transacción de negocio se ha ejecutado satisfactoriamente.

Función de LoadRunner que hay que usar**: web_reg_find()**

```
int web_reg_find (const char *attribute_list, LAST)
```

Descripción:

Permite buscar un *String* dentro de una página HTML

<NOTA>: Hay que ejecutarla ANTES de solicitar la página al Servidor

Return:

0 **Si encuentra el *String***

1 **Si no encuentra el *String***
Provoca el Error Nº -26366 en tiempo de ejecución. Ejemplo:

```
Error -26366: vuser_init.c(32) Error -26366 "Text=ccccc" not found for web_reg_find
```

Ejemplo: comprobar que tras enviar un formulario de consulta del DNI 11111111 al servidor, la página se ha ejecutado correctamente y devuelve el contenido solicitado

web_reg_find("Text=Consulta realizada correctamente", LAST);

web_reg_find("Text= 11111111", LAST);

```
web_submit_form("login.pl", "Snapshot=t2.inf",
        ITEMDATA,
        "Name=username", "Value=jojo", ENDITEM,
        "Name=password", "Value=bean", ENDITEM,
        "Name=login.x", "Value=35", ENDITEM,
        "Name=login.y", "Value=14", ENDITEM,
        LAST);
```

En el segundo chequeo introducido el valor del DNI debería estar parametrizado con uno de los parámetros de entrada, para verificar que la aflicción ha devuelto la consulta del DNI solicitado. La sentencia quedaría de la siguiente manera:

web_reg_find("Text= {dpDNI}", LAST);

Todas las peticiones al servidor deben tener puntos de chequeo de contenido que aseguren que la transacción se ha ejecutado correctamente y devuelve lo esperado.

3.1.5 Chequeos de contenido con WebServices

Para realizar una verificación del contenido devuelto por una llamada WebServices se debe realizar de la siguiente manera:

```
char* prmStrVariable;

lr_start_transaction("TRAN_XXX_001_YYY");

soap_request("StepName=SOAP Request",
        "URL={dpURL}",
        "SOAPEnvelope={dpSolicitud}",
        "SOAPAction=XXXXXXXX",
        "ResponseParam=prmResultado",
        LAST);

prmStrVariable= lr_eval_string("{prmResultado}");

if (strstr(prmStrVariable ,"OK"))
    {
  lr_output_message( "Correcto");
    }
else
    {
  lr_fail_trans_with_error("Error");
    }

lr_end_transaction("TRAN_XXX_001_YYY", LR_AUTO);
```

En el parámetro ‘prmResultado’ se recupera la respuesta de llamada SOAP y se comparará dicho resultado con el esperado (en el caso del ejemplo sería “OK”). En caso de que coincidan, se muestra un mensaje de “Correcto” (opcional) y si no coinciden se hace que la transacción falle y muestre un mensaje de error.

Todas las peticiones al servidor deben tener puntos de chequeo de contenido que aseguren que la transacción se ha ejecutado correctamente y devuelve lo esperado.

3.1.6 Mensajes

En grandes ejecuciones o mantenidas en el tiempo hay que evitar mensajes del tipo:

```
lr_vuser_status_message("dpIteration: %s - START Action", lr_eval_string("{dpIteration}"));
lr_output_message("dpIteration: %s - START Action", lr_eval_string("{dpIteration}"));
```

Ejecutados por cada iteración impactan de forma negativa en el rendimiento ya que aumenta el tráfico y el Controller tiene que gestionar todos los mensajes de todos los Vusers, lo que provoca que se reduzca dramáticamente la escalabilidad del Controller.

Generalmente se considera una mala práctica insertar **lr_error_message** en los scripts y por dicho motivo **no deberán ser utilizados** en los scripts que se entreguen a la oficina de Pruebas.

3.1.7 Resultados

Cuando se realizan varias iteraciones en la ejecución de un script se generan unos ficheros de resultados que pueden ocupar bastante espacio (ejemplo: 500 Mb). Para evitar que se generen estos resultamos y se suban a ALM debemos realizar las siguientes acciones.

En el Vugen:

1. Ir al menú “**Tools /Options”.**
2. En la sección **Scripting**, en la pestaña **Replay** observar que no esta marcado el check de ‘**Results / Enable results of replay summary to be saved to a named folder after each script run’**.

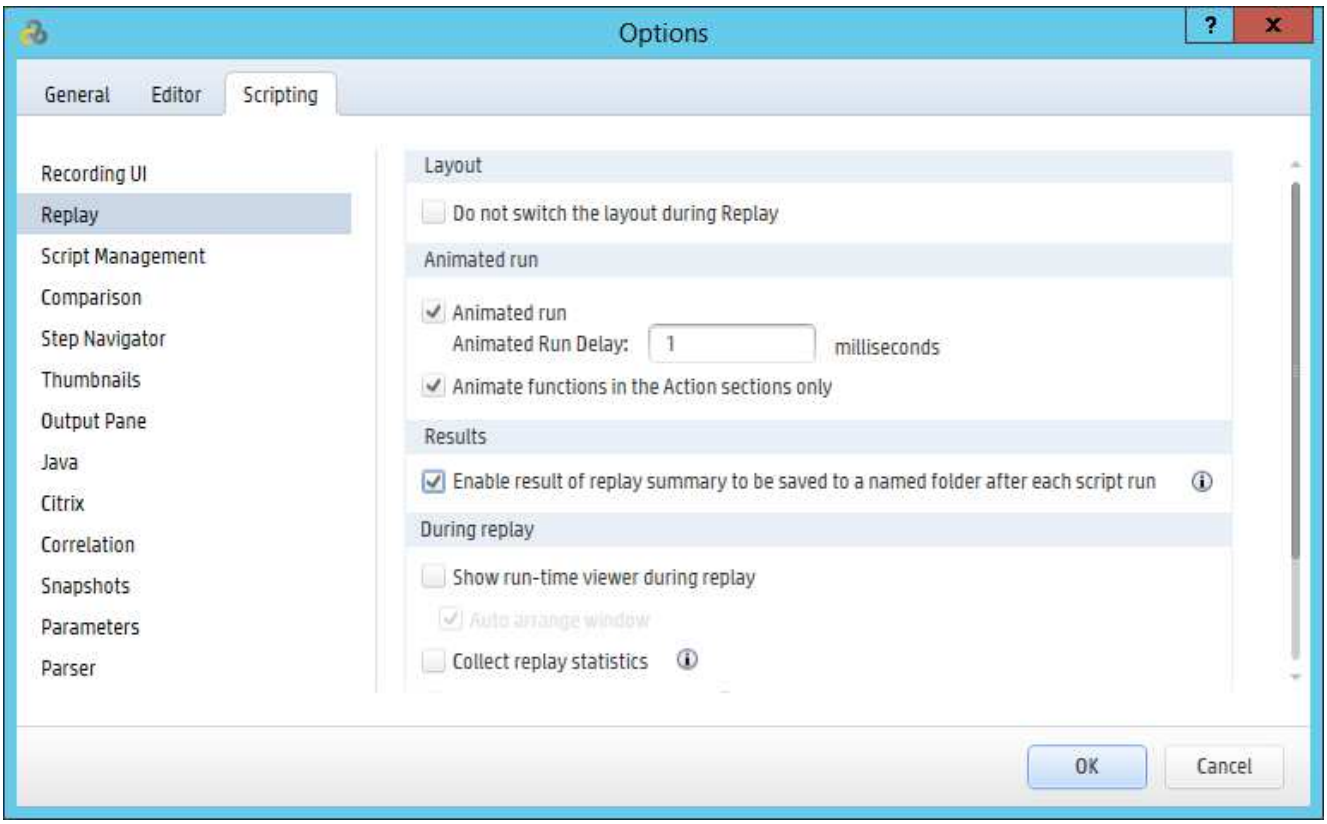


Ilustración 29. Opciones de configuración

Si los ficheros de resultados ya se han generado lo que deberemos realizar son las siguientes acciones.

1. Ir al directorio temporal del script.
2. Seleccionar la carpeta “result1”.

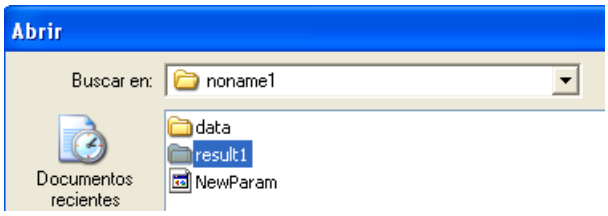


Ilustración 30. Directorio de resultados

3. Pulsar en el botón eliminar.

3.1.8 Generation Log y Recording Log

Durante la grabación del proceso de negocio se generan los logs Generation Log y Recording Log que guardan información sobre la sesión de grabación como peticiones al servidor, respuestas del mismo, cookies, snapshots, datos, opciones de grabación, etc que son necesarios para la validación de los scripts y para la solución de posibles errores de reproducción en los mismos.

3.2 Normas adicionales

3.2.1 Protocolos: Web (HTTP/HTML) o Web Services

Se disponen de licencias para los protocolos **Web (HTTP/HTML) o WebService**, por lo que son los únicos con los que se puede grabar los scripts:

Web (HTTP/HTML)

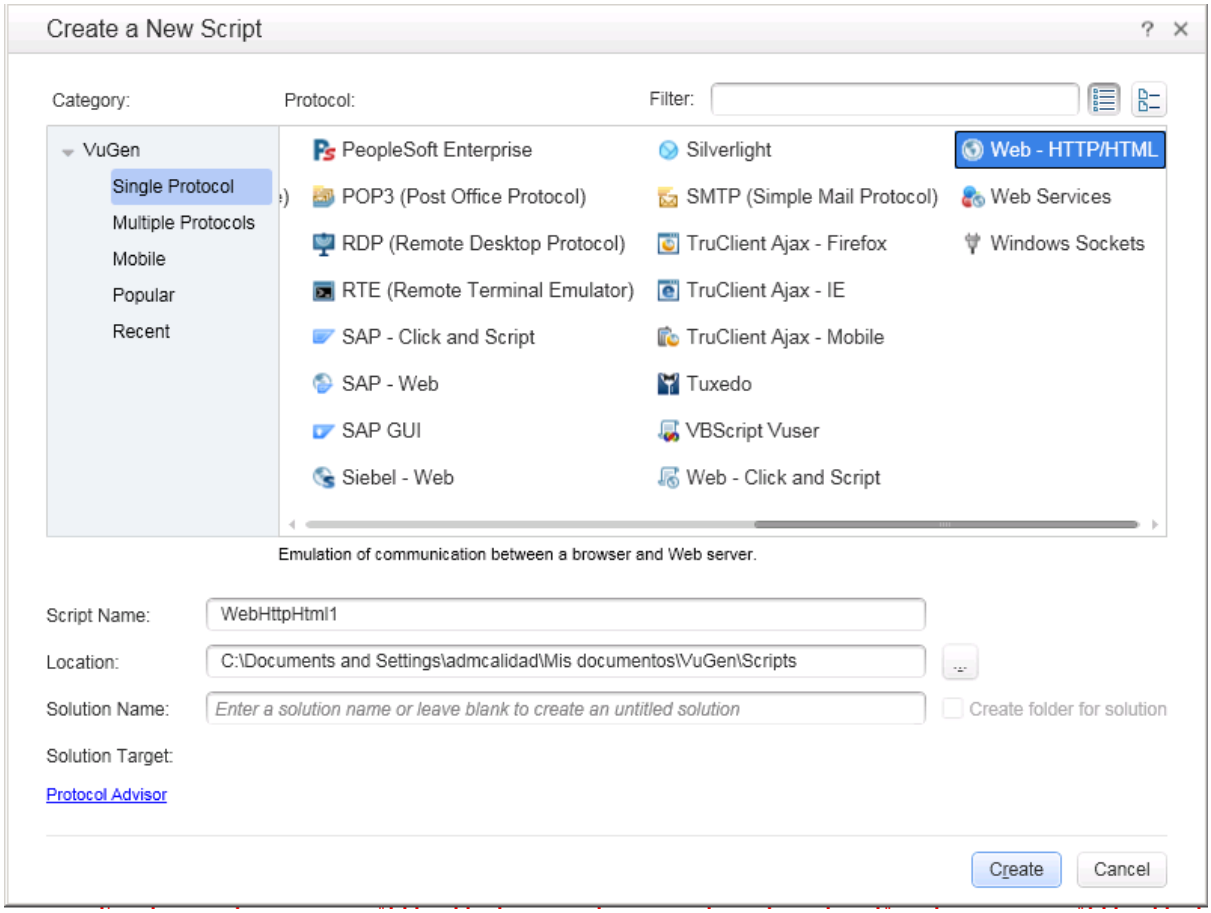


Ilustración 31. Ventana selección de protocolos I

WebService

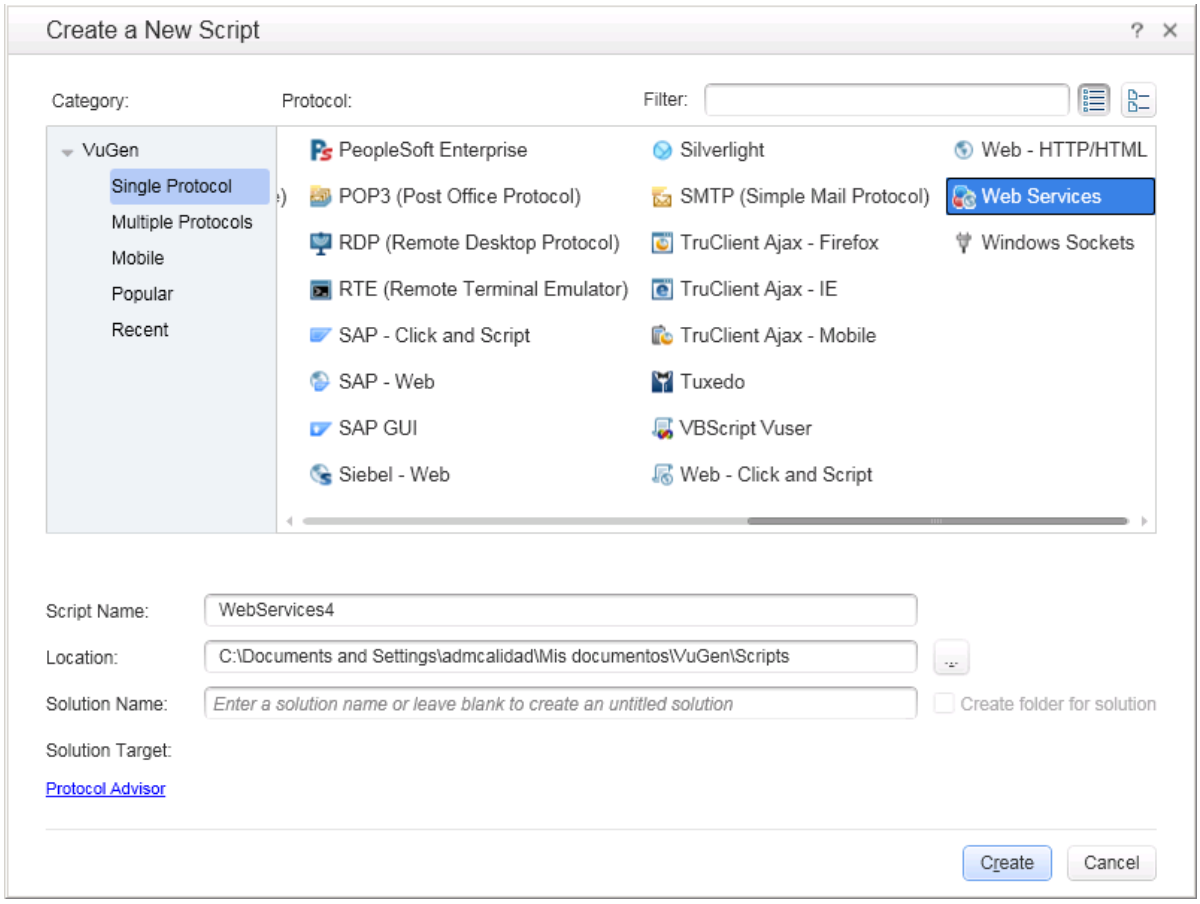


Ilustración 32. Ventana selección de protocolos II

3.2.1.1 Caso: Chequeo en la descarga de un fichero

3.2.1.1.1 Descripción

En algunas ocasiones al realizar una navegación web nos generará un fichero para descargar, por ejemplo un pdf. En dichos casos no se deberá pulsar al botón de abrir o guardar fichero, la comprobación que se realizará será sobre el tamaño del fichero.

Solución: Utilizar web_get_int_property()

La función **web_get_int_property()** devuelve información específica sobre la petición HTTP anterior.

3.2.1.1.1.1 Sintaxis

```
int web_get_int_property( const int HttpInfoType );
```

3.2.1.1.1.2 Ejemplo

```
int prmFileSize;

web_url("HTTP Request",
        "URL=http://{dpURL}",
        "TargetFrame=_TOP",
        LAST );

prmFileSize = web_get_int_property(HTTP_INFO_DOWNLOAD_SIZE);
// HTTP_INFO_DOWNLOAD_SIZE devuelve el tamao en bytes de la descarga.

if (prmFileSize < 10000)
{
        lr_end_transaction("TRAN_XXX_001_YYY", LR_FAIL);
}
else
{
        lr_output_message("El script ha accedido correctamente a la pgina especificada");
        lr_end_transaction("TRAN_XXX_001_YYY", LR_PASS);
}

return 0;
```

3.2.1.2 Caso: Cómo terminar la ejecución de un Usuario Virtual de forma ordenada

3.2.1.2.1 Descripción

En algunas ocasiones necesitamos terminar la ejecución de la actividad de un Usuario Virtual de forma ordenada. Por ejemplo, cuando no tengamos más datos que procesar, cuando se cumpla una condición, cuando se finalice un determinado número de iteraciones, cuando no haya más datos en un fichero de datos de entrada, etc…

3.2.1.2.2 Solución: Utilizar la Función lr_exit()

La función **lr_exit()** termina la ejecución de un script, acción o iteración.

3.2.1.2.2.1 Sintaxis

```
void lr_exit (int continuation_option, int exit_status);
```

Donde:

`continuation_option`	**Especifica cómo continuará la ejecución del Script** Ver los posibles valores en la tabla de abajo
`exit_status`	**Es el Exit Status del Grupo al que pertenece el Script dentro de un Escenario de la Herramienta Controller** **Ver los posibles valores en la tabla de abajo**

3.2.1.2.2.2 Posibles valores de los argumentos

Argumento: **continuation_option**

C Constants	Object Oriented Constants	Behavior
LR_EXIT_VUSER	lr.EXIT_VUSER	Exit without any condition, and go directly to end action
LR_EXIT_ACTION_ AND_CONTINUE	lr.EXIT_ACTION_ AND_CONTINUE	Stop current action, and go to the next action.
LR_EXIT_MAIN_ ITERATION_ AND_CONTINUE	lr.EXIT_MAIN_ ITERATION_ AND_CONTINUE	Stop current global script run iteration, and go to the next iteration.
LR_EXIT_ITERATION_ AND_CONTINUE	lr.EXIT_ITERATION_ AND_CONTINUE	Stop current iteration, and go to the next iteration. if called from within a block iteration, only the block iteration will be exited, and not the global iteration.
LR_EXIT_VUSER_	lr.EXIT_VUSER_	Run until the end of the current iteration and

AFTER_ITERATION	AFTER_ITERATION	then exit.
LR_EXIT_VUSER_ AFTER_ACTION	lr.EXIT_VUSER_ AFTER_ACTION	Run until the end of the current action and then exit.

Tabla 1. Valores continuation_option de lr_exit

Argumento: **exit_status**

C Constants	Object Oriented Constants
LR_PASS	lr.PASS
LR_FAIL	lr.FAIL
LR_AUTO	lr.AUTO

Tabla 2. Valores exit_status de lr_exit

<NOTA>: Para terminar la ejecución del Script de forma ordenada, se debe realizar la siguiente llamada:

```
lr_exit (LR_EXIT_VUSER , LR_PASS);
```

3.2.1.2.2.3 Ejemplo

En este ejemplo, ejecutamos un "*submit*" de un Formulario Web. Si no hay datos en el Formulario, terminaremos la ejecución del Usuario Virtual de forma ordenada, es decir, el script finaliza y no da ningún error.

```
…
if (atoi(lr_eval_string("{dpExistData}")) == 1)      // Comprobamos si hay registros
{                                                    // en el Formulario Web, es decir
                                                     // si hay datos para procesar
```

```
        /*
        ------------------------------------------------------------
        HAY DATOS EN EL FORMULARIO, POR LO TANTO, HACEMOS EL SUBMIT:
        ------------------------------------------------------------
        */
        lr_start_transaction("TRAN_APLI_002_Validar_Expediente");
        web_submit_data("DGTListadoPendientesPaginado.do",
                "Action=http://{dpUrl}/EPVServidor/DGTListadoPendientesPaginado.do",
                "Method=POST",
                "RecContentType=text/html",
                "Referer=http://{dpUrl}/EPVServidor/DGTFiltroListadoPendientesPaginado.do",
                "Snapshot=t15.inf",
                "Mode=HTML",
                ITEMDATA,
                "Name=tipoListado", "Value=TIPO_VALIDACION", ENDITEM,
                "Name=fechaDesde", "Value=", ENDITEM,
                "Name=fechaHasta", "Value=", ENDITEM,
                "Name=opcionFiltro", "Value=OPCION_NO_PARALIZADOS", ENDITEM,
                "Name=obj[0].numero", "Value=on", ENDITEM,
                LAST);
        lr_end_transaction("TRAN_APLI_002_Validar_Expediente",LR_AUTO)
}
else
{
        /*
        ------------------------------------------------------------
        NO HAY MAS DATOS EN EL FORMULARIO, POR LO TANTO,
        TERMINAMOS EL SCRIPT DE MANERA ORDENADA:
        LLAMAMOS A LA FUNCION lr_exit() PASANDOLE POR PARAMETRO
        LOS ARGUMENTOS LR_EXIT_VUSER (TERMINAR SIN NINGUNA CONDICION
        Y LR_PASS (DAMOS POR OK LA EJECUCION DE LA TRANSACCION):
        ------------------------------------------------------------
        */
        lr_exit (LR_EXIT_VUSER , LR_PASS);
}
return 0;
```

3.2.1.3 Caso: Uso de Librerías DLL externas

3.2.1.3.1 Descripción

LoadRunner incluye una gran cantidad de Funciones propias, a través de la cuales podremos implementar multitud de funcionalidades. Asimismo, podemos ejecutar todas las Funciones ANSI-C. Si aun así, necesitamos funcionalidades adicionales (encriptar y desencriptar *Strings*, conectarnos a Bases de Datos, etc …), podemos ejecutar DLLs externas, tanto de fabricantes de SW como creadas por nosotros mismos.

<NOTA>: Sólo podemos llamar a **DLLs "puras"**, es decir, a DLLs que tienen punto de entrada (*entry point*) y sus Funciones están exportadas.
Las DLLs ActiveX generadas con Visual Basic **NO funcionan**.

3.2.1.3.2 Solución: Utilizar la Función lr_load_dll()

La función **lr_load_dll()** carga una librería DLL en tiempo de ejecución. Una vez cargada, podremos llamar a sus Funciones como si se trataran de Funciones "normales" dentro del script.

3.2.1.3.2.1 Sintaxis

```
int lr_load_dll (const char *library_name);
```

Donde:

`library_name` **Nombre de la DLL**

<NOTA>: Tenemos que instalar la librería DLL en el Directorio **`system32`** de Windows o en un Directorio incluido en la variable de entorno `PATH`

3.2.1.3.2.2 Ejemplo

Queremos decodificar un *String* que está codificado en base64. Como LoadRunner no tiene ninguna función propia que nos permita realizar esta acción, vamos a utilizar una librería DLL externa: base64.dll.

Pasos a realizar:

1. Copiar la librería base64.dll en el Directorio system32 de Windows o en un Directorio que esté incluido en la Variable PATH de la máquina donde tenemos el script de LoadRunner
2. Desde el script de LoadRunner realizar lo siguiente:
   - Dentro de la Transacción vuser_init, cargar la DLL en tiempo de ejecución:

     lr_load_dll("base64.dll");

   - Llamar a la función **Base64DecodeA** de la DLL **base64.dll**, que nos permite desencriptar un *String* que está codificado en base64:

     Base64DecodeA (prmStrEncoded, prmStrDecoded, BytesNeededToDecode(strlen(**prmStrEncoded**), CountEqualSignsA(**prmStrEncoded**)));

prmStrEncoded es el *String* codificado en base64 y en la variable **prmStrDecoded** se almacenará el *String* decodificado

3.2.1.4 Caso: Crear Ficheros de Datos de Entrada dinámicamente, a través de la ejecución de Scripts SQL

3.2.1.4.1 Descripción

Hay casos en los que los Datos de Entrada necesarios para la ejecución de las Pruebas han de crearse en tiempo de ejecución. Es decir, antes o durante la ejecución de una Prueba, tenemos que crear los ficheros de datos de entrada (cuyo formato de nombre es DENT_<Aplicación>_<Entorno>…)

La solución más óptima es diseñar scripts SQL externos al script de LoadRunner que permitan conectarse a la Base de Datos origen, y ejecutar las sentencias SQL necesarias que creen los ficheros de datos de entrada.

Estos scripts externos son ficheros de Shell script (.bat en Windows y .sh en UNIX) que invocan a scripts de SQL (ficheros .sql) que ejecutan las sentencias SQL necesarias para crear los ficheros de datos de entrada.

La llamada a estos scripts externos se podrá realizar:

- Desde el script de LoadRunner, a través de la función **system()** de C
- Directamente desde la línea de comandos del Sistema Operativo
- A través de un *Scheduler* de procesos ("Tareas programadas de Windows", CONTROL-M, comando `crontab` de UNIX, etc…)
- Desde el Módulo "*Test Lab*" de ALM

<NOTA>: En lugar de realizar llamadas a Scripts externos, podemos incluir sentencias SQL embebidas directamente dentro de los Scripts de **LoadRunner**
Dependiendo del caso en concreto y el objetivo, tendremos que valorar cuál es la mejor técnica a seguir. Aquí sólo describiremos cómo llamar a Scripts SQL externos.

3.2.1.4.2 Solución

En la siguiente imagen se detallan las diferentes vías a través de las cuales podemos ejecutar el Shell Script y cómo se generan los ficheros de datos de entrada **dinámicamente**:

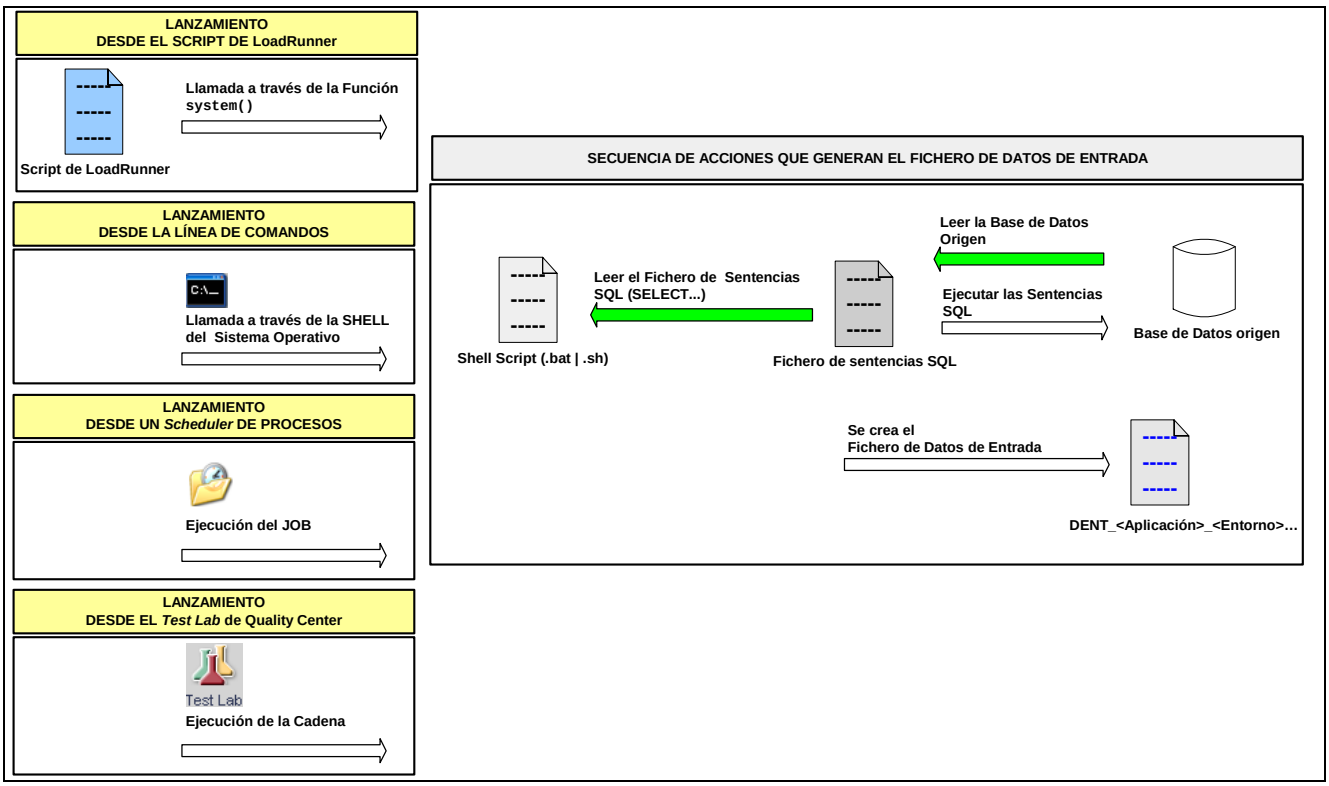


Ilustración 33. Diagrama de secuencia de generación datos

3.2.1.4.2.1 Ejemplo

Vamos a lanzar una Prueba de Carga de la Aplicación PEPV. El script que vamos a lanzar, lee un fichero de datos de entrada que contiene los valores del campo: “SICER” de la tabla: “documentos_gen_notif_acuse” . Vamos a generar dinámicamente el fichero de datos de entrada, ejecutando un Shell Script.

3.2.1.4.2.2 Dónde se instalan los Shell Scripts y Ficheros .sql en la estructura de Directorios de montaje

En el apartado: “**Estructura de Directorios de montaje**” se detalla dónde se alojan los Shell Scripts y los ficheros de sentencias SQL, organizados por Aplicación y versión.

En el ejemplo que nos ocupa, los ficheros implicados son los siguientes:

Directorio	Ejemplo	Descripción

bin	**launch_step_002.bat**	Lanza la ejecución del Shell Script: **select_sicer_001.bat**
bin	**select_sicer_001.bat**	Lanza la ejecución del Fichero .sql: **select_sicer_001.sql**
sql	**select_sicer_001.sql**	Se conecta la BBDDs de la Aplicación PEPV, ejecuta una SELECT para recuperar el Campo: "**SICER**" de la Tabla: **documentos_gen_notif_acuse** y vuelca el resultado en el Fichero de Datos de Entrada: **DENT_PEPV_PRE_001.dat**

Tabla 3. Ficheros para la generación dinámica de datos

3.2.1.4.2.3 Lanzamiento

Debemos ejecutar el Shell Script: launch_step_002.bat, antes del lanzamiento de la Prueba de Carga. El fichero de datos de entrada que se generará es: DENT_PEPV_PRE_001.dat

3.2.1.5 Caso: Existe información dinámica que se genera dependiendo de los parámetros introducidos, para estos casos se utilizará la Correlación.

3.2.1.5.1 Descripción

Existen casos en los que tras la grabación de un script de carga **Vugen** con unos parámetros de entrada determinados se genera información dinámica necesaria para el resto de la navegación. Al intentar reproducir el script nos dará error ya que esa información es dinámica y se genera en tiempo de ejecución.

Un ejemplo ilustrativo podría ser en una Web de Compra de Billetes de Avión. Al parametrizar las ciudades origen y destino de los vuelos, el identificador del vuelo varía. Este identificador es necesario para finalizar la compra del billete de avión. Igual ocurre en aplicaciones que llevan un identificador de sesión. Este puede variar cada vez que nos conectemos a la Web.

3.2.1.5.2 Solución

La solución en estos casos es la **Correlación**, que nos permite guardar automáticamente los datos dinámicos generados y usarlos en las sucesivas navegaciones.

3.2.1.5.2.1 Ejemplo

Una vez grabado nuestro script con la herramienta **Vugen** realizaremos las siguientes acciones.

1. Ir a "**Design/Design Studio**"

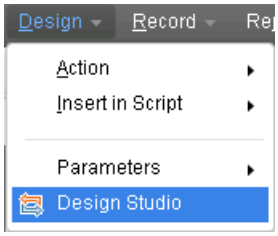


Ilustración 34. Opción de menú

2. Pulsar "**Replay&Scan**" en la ventana que aparece.

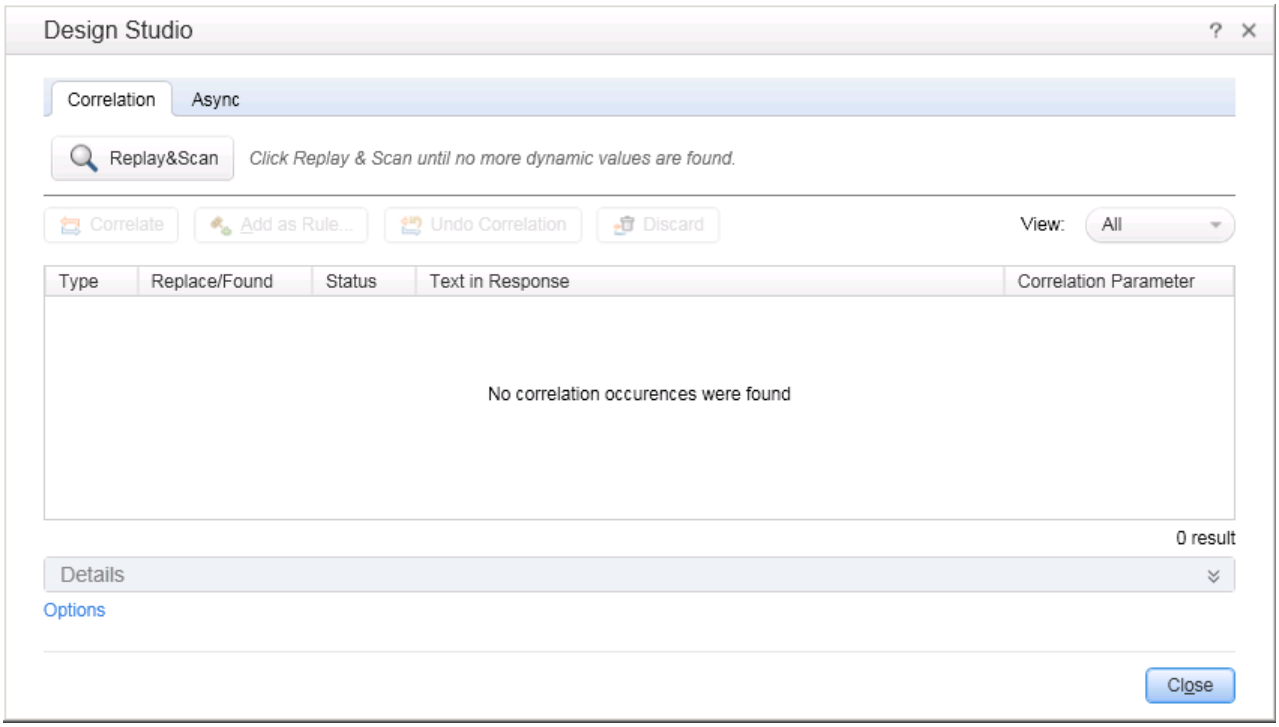


Ilustración 35. Ventana de correlación

3. En la parte de abajo de la pestaña "**Correlation Result**" seleccionar cada una de las correlaciones que aparecen y pulsar en el botón "**Correlate**".

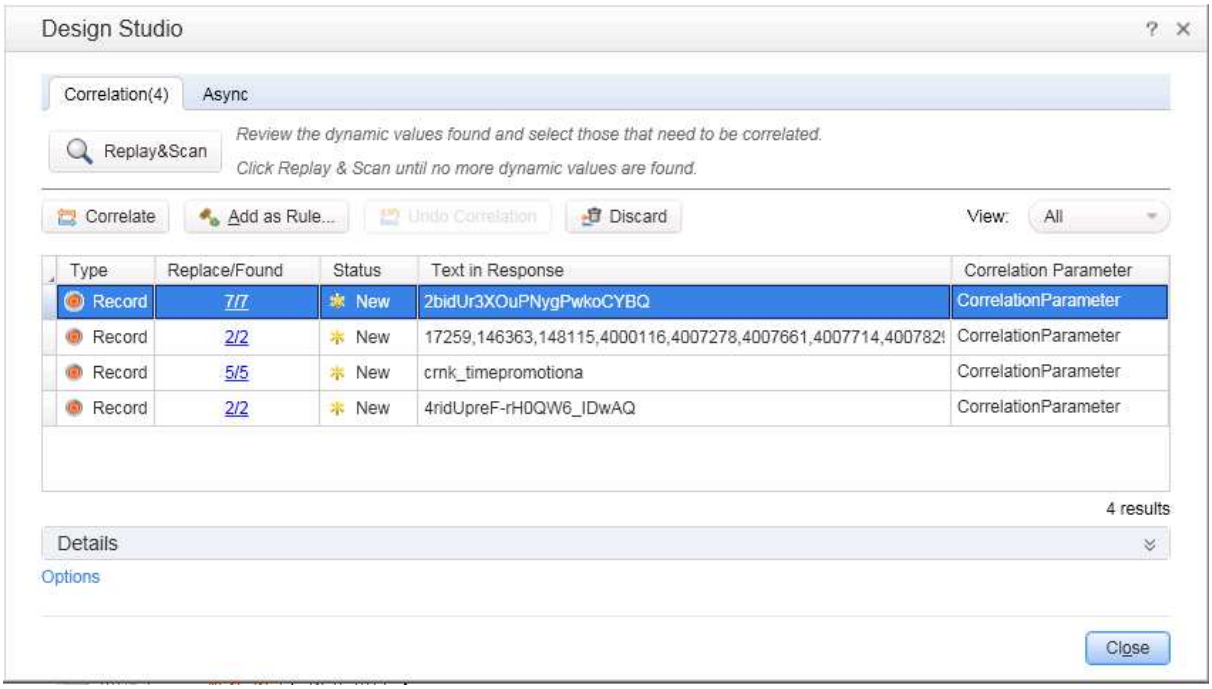


Ilustración 36. Valores identificados de correlación

4. Al finalizar pulsar en "**Replay**".

3.2.1.6 Caso: Scripts que utilizan archivos locales en su grabación, para estos casos se utilizará Add Files.

3.2.1.6.1 Descripción

Existen casos en los que durante la grabación de un script de carga **Vugen** se utilizan ficheros locales como datos en el proceso de negocio. Una vez subidos a ALM, al volver a descargar el script a local (bien desde el **Vugen** para probar, bien desde el **Controller**) los ficheros utilizados no aparecen, por lo que Loadrunner utiliza un fichero vacío de tamaño 0 y falla la ejecución del script.

Un ejemplo ilustrativo podría ser la carga de un archivo de imagen o un pdf en la aplicación.

3.2.1.6.2 Solución

La solución en estos casos es añadir los fichero utilizados mediante la opción Add files to Script….

Para ello pulsaremos botón derecho en "**Extra Files**", de la ventana "**Solution Explorer**":

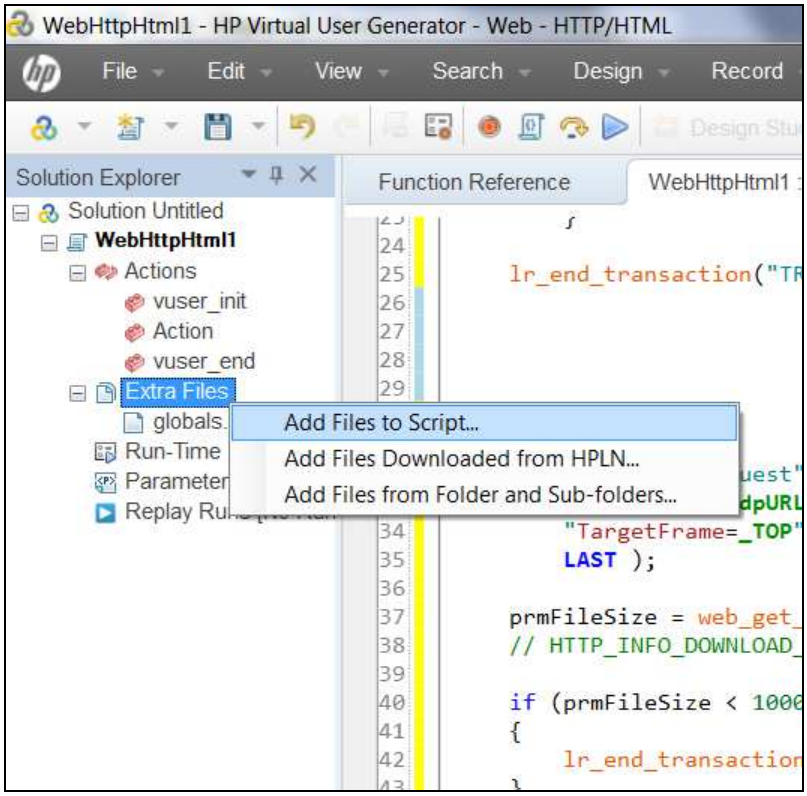


Ilustración 37. Add extra files

Posteriormente se debe guardar el script en ALM.

3.2.1.7 Caso: Añadir Cabeceras HTTP en el Script

3.2.1.7.1 Descripción

Si queremos enviar “instrucciones adicionales” al Servidor Web, tenemos que utilizar Cabeceras HTTP. Por ejemplo, si tenemos que enviar el Login y Password de un Usuario **antes** de ejecutar un WebService, tenemos que hacerlo con Cabeceras HTTP.

3.2.1.7.2 Solución: Utilizar la Función web_add_header()

Permite enviar una Cabecera HTTP. Se envía el nombre de la Cabecera y su valor.

3.2.1.7.2.1 Sintaxis

```
int web_add_header (const char *Header, const char *Content);
```

Donde:

`Header`	**Nombre de la Cabcera**
`Content`	**Valor de la Cabecera**

3.2.1.7.2.2 Ejemplo

Vamos a ejecutar un WebService con autentificación de Usuario. **Antes** de ejecutar el WebService, debemos enviar el Login y el Password al Servidor HTTP.

Tenemos que enviar las Cabeceras `"Authorization:"` y `"SOAPAction:"`.

En la Cabecera "**`Authorization`**:", enviamos el Login y Password del Usuario en formato:

login/password

y codificado en Base64. El valor de esta Cabecera es : "**`Basic MTIzNDU2Nzg6REdUNjc4`**"

El Código Fuente de este ejemplo es el siguiente:

```
/*
-----------------------------------------------------------
ENVIAMOS EL LOGIN Y PASSWORD A TRAVÉS DE CABECERAS HTTP:
-----------------------------------------------------------
*/

web_add_header("Authorization:", "Basic MTIzNDU2Nzg6REdUNjc4");

web_add_header("SOAPAction:", "\"");
/*
-----------------------------------------------------------
LLAMAMOS AL WebService A TRAVÉS DE LA FUNCIÓN soap_request():
-----------------------------------------------------------
*/

soap_request("StepName=Step_001",
"URL=http://{dpURL}/ConsultaVehiculos/services/ConsultaAntecedentesVehiculos",
"SOAPEnvelope=<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
…
```

```
"\r\n</soapenv:Envelope>\r\n",
"Snapshot=t1.inf",
"ResponseParam=prmResult",
LAST);
```

3.2.2 Protocolo: HTTP/HTML

3.2.2.1 Caso: Cómo grabar Scripts https con Fichero de Certificado Digital

3.2.2.1.1 Descripción

Si la URL de la Aplicación que queremos implementar es de tipo https, entonces tenemos que instalar el fichero de certificado digital en el navegador **Internet Explorer** de las siguientes máquinas:

- En la máquina en la que se graba el script.
- En las máquinas en las que se ejecuta la prueba, es decir en los Inyectores de Carga ("**Load Generators**").

y seguir los pasos que se describen en el siguiente apartado.

3.2.2.1.2 Solución

<NOTA>: Los responsables de la Aplicación nos tienen que proporcionar todos los Ficheros de Certificados Digitales https y sus Passwords

Realizar los siguientes pasos:

1. Instalar los ficheros de certificados digitales https de la aplicación en el navegador Internet Explorer, que es el standard de la Plataforma de Sistema Operativo de la DGT
2. Ejecutar la herramienta **LoadRunner VuGen** y crear un nuevo script de Protocolo HTTP/HTML
3. Ejecutar la opción: "**Record -> Recording Options -> Network -> Port Mapping**" y seleccionar la opción "**WinINet level data**", como se indica en la siguiente imagen:

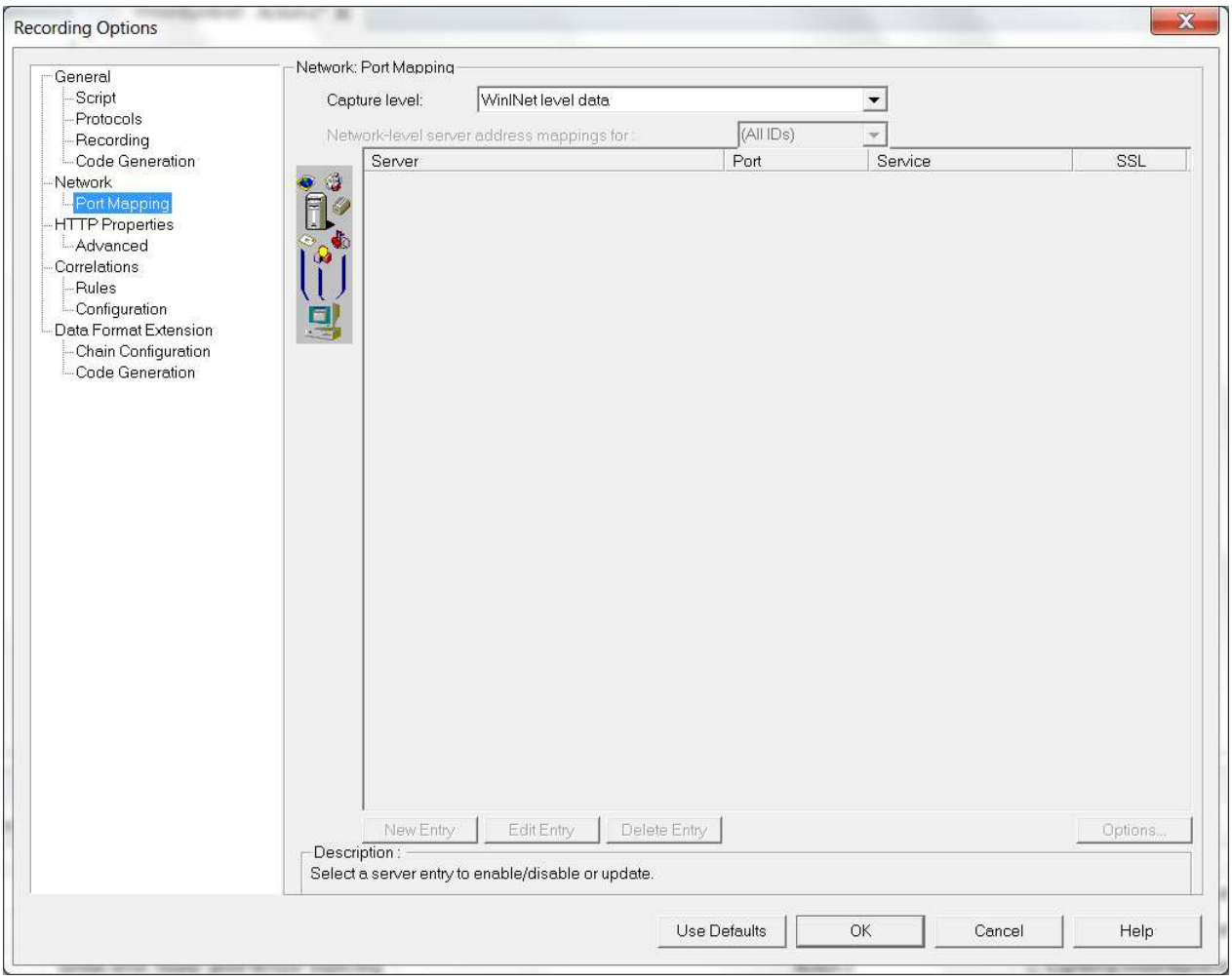


Ilustración 38. Recording options. Port mapping

4. Pulsar el botón: "**OK**"
5. En el Solution Explorer, abrir : "**Solution Explorer → Run-Time Settings → Internet Protocol → Preferences**"
6. En el apartado "**Advanced**", comprobar que está deshabilitada la opción "**WinInet replay instead of Sockets**":

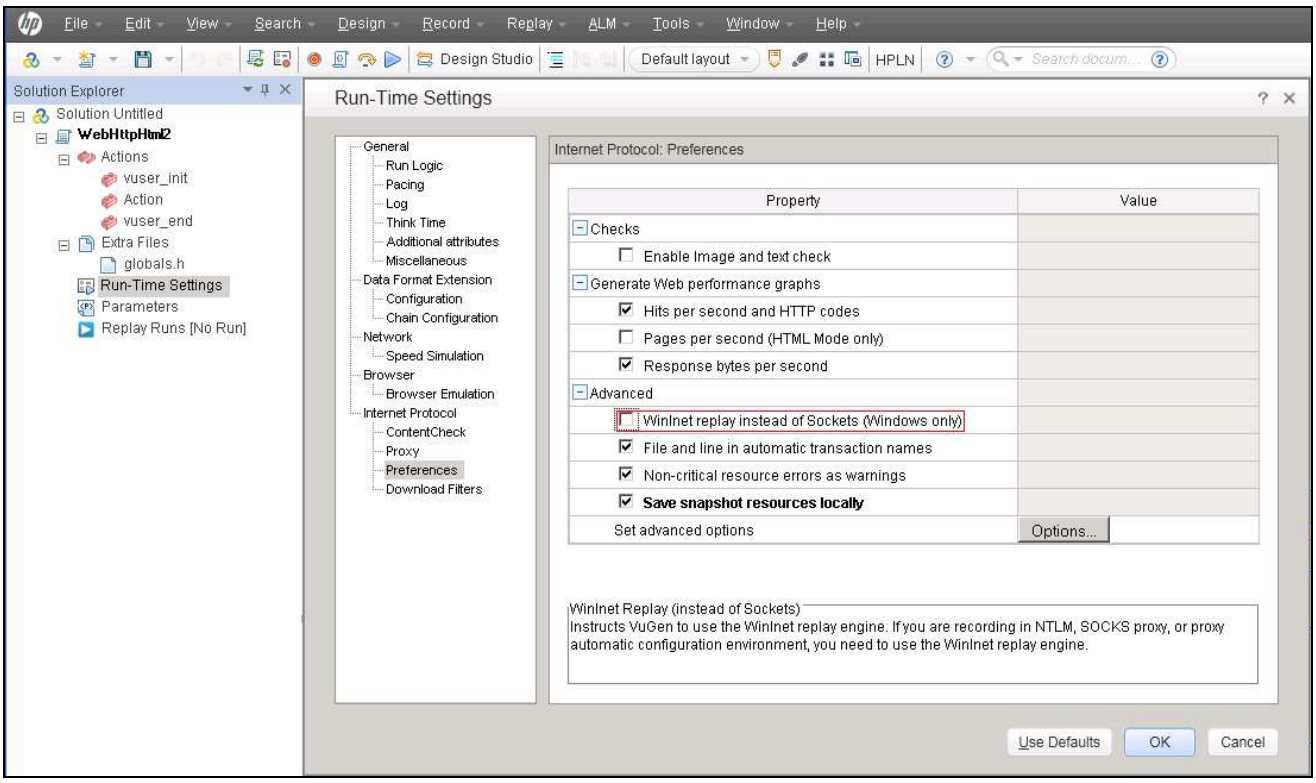


Ilustración 39. Ventana de Run-time settings

7. Grabar el script.

Si observamos el código fuente del script que se ha generado después de la grabación, podemos comprobar que se ha introducido una llamada a la función: **web_set_certificate_ex()**, la cual establece el fichero del certificado digital https que se ha utilizado.

Este Fichero es uno de los que hemos insertado en el navegador **Internet Explorer** en el Paso 1

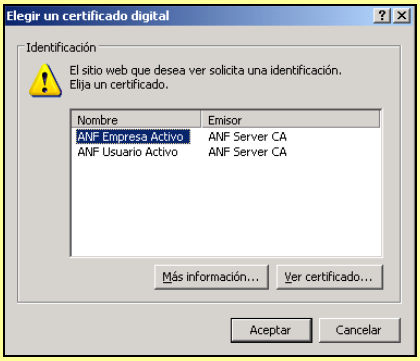
Un ejemplo de llamada a esta función es el siguiente:

```
web_set_certificate_ex("CertFilePath=WinINetCert1.pem",
                        "CertFormat=PEM",
                       "KeyFilePath=WinINetCert1.pem",
                       "KeyFormat=PEM",
                       "Password=4746c4d4",
                       "CertIndex=1",
                       LAST);
```

Podemos parametrizar los valores de los argumentos indicados en **negrita**, es decir, el nombre del fichero del certificado digital https (**`CertFilePath`** y **`KeyFilePath`**) y el Password (**`Password`**) contra un fichero de datos de entrada.

<NOTA>: El Password del Certificado Digital está encriptado

<NOTA>: Cuando ejecutemos el Script **desde LoadRunner VuGen** nos saldrá la ventana de Internet Explorer para seleccionar el Certificado Digital a utilizar. Por ejemplo, una ventana similar a ésta:



Seleccionamos el Certificado y seguimos con la ejecución del Script.

Cuando ejecutemos el Script desde **LoadRunner Controller** **NO** saldrá esta ventana.

3.2.2.2 Caso: Realizar búsquedas especiales en las llamadas a la Función web_reg_find()

3.2.2.2.1 Descripción

La función **web_reg_find()**, que permite buscar un String dentro del Código Fuente de una Página Web, admite opciones avanzadas de búsqueda.

3.2.2.2.2 Solución

3.2.2.2.2.1 Método 1: Desde el asistente:

1. Abrir **"View/Step Navigator".**

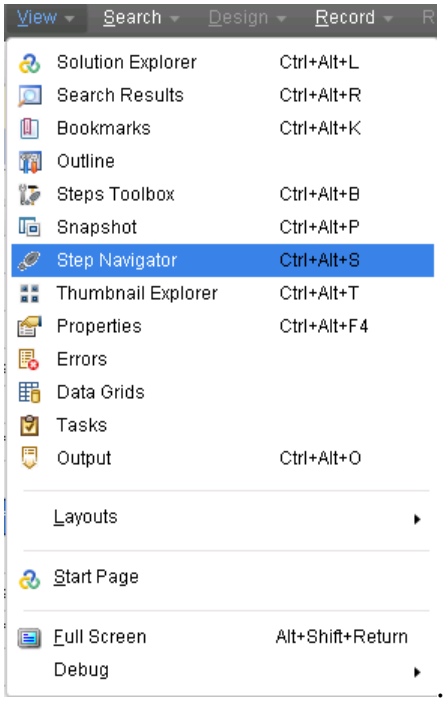


Ilustración 40. Opciones menú View

2. En el **Step Navigator**, buscar el paso **Submit Form** a comprobar.

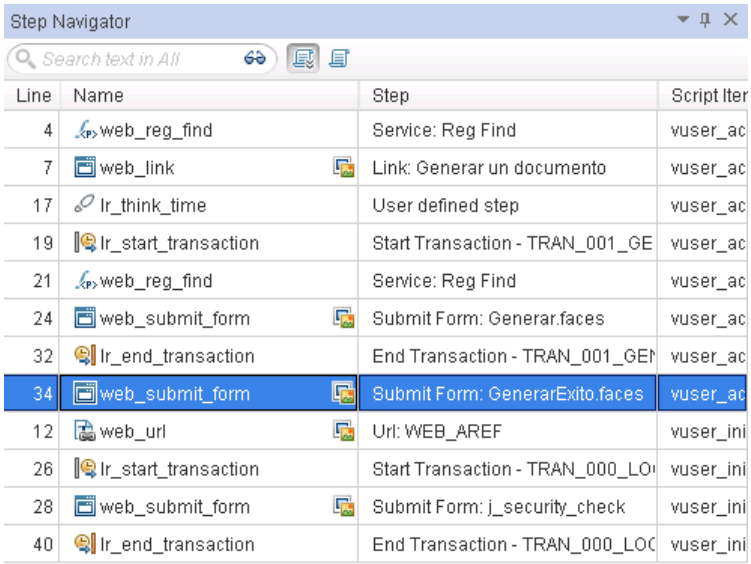


Ilustración 41. Step navigator

3. En la barra de tareas, pulsar el botón **Snapshot** para mostrar el panel de **Snapshots**.

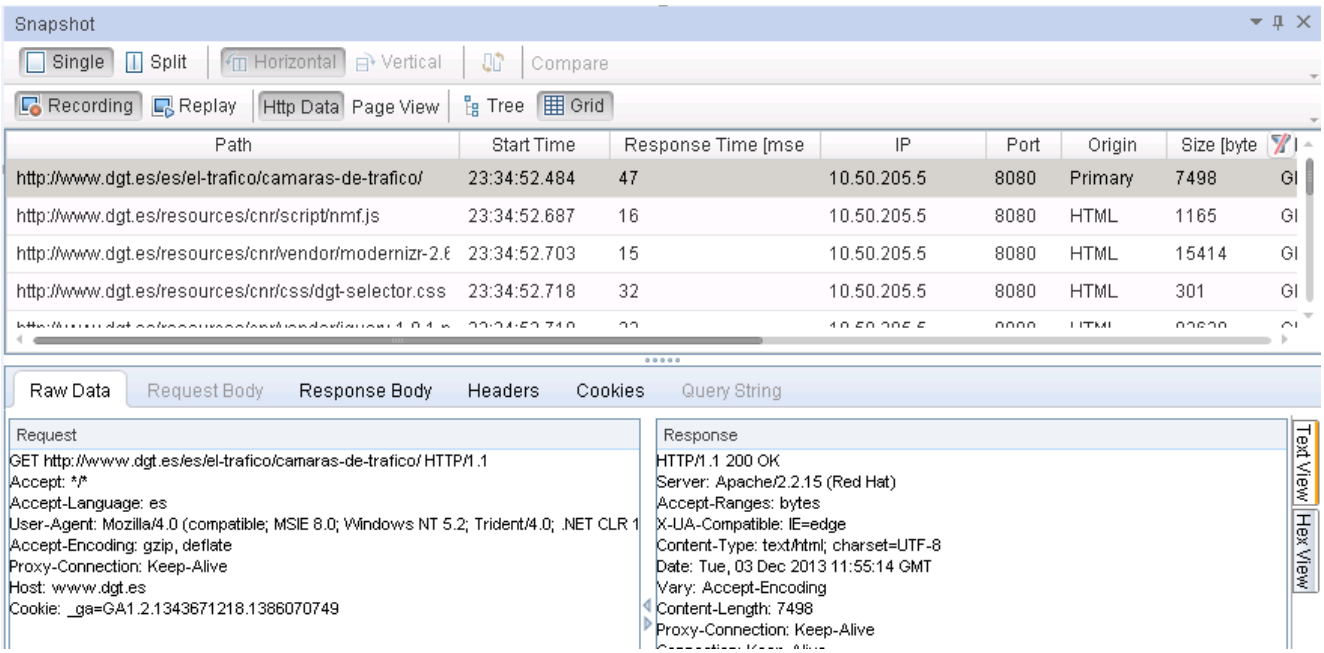


Ilustración 42. Snapshots

4. En el **Step Navigator**, hacer doble click en el icono de snapshot en el paso que queramos comprobar. El snapshot asociado aparecerá en el panel de Snapshot y se mostrará la línea del script asociada a la petición.

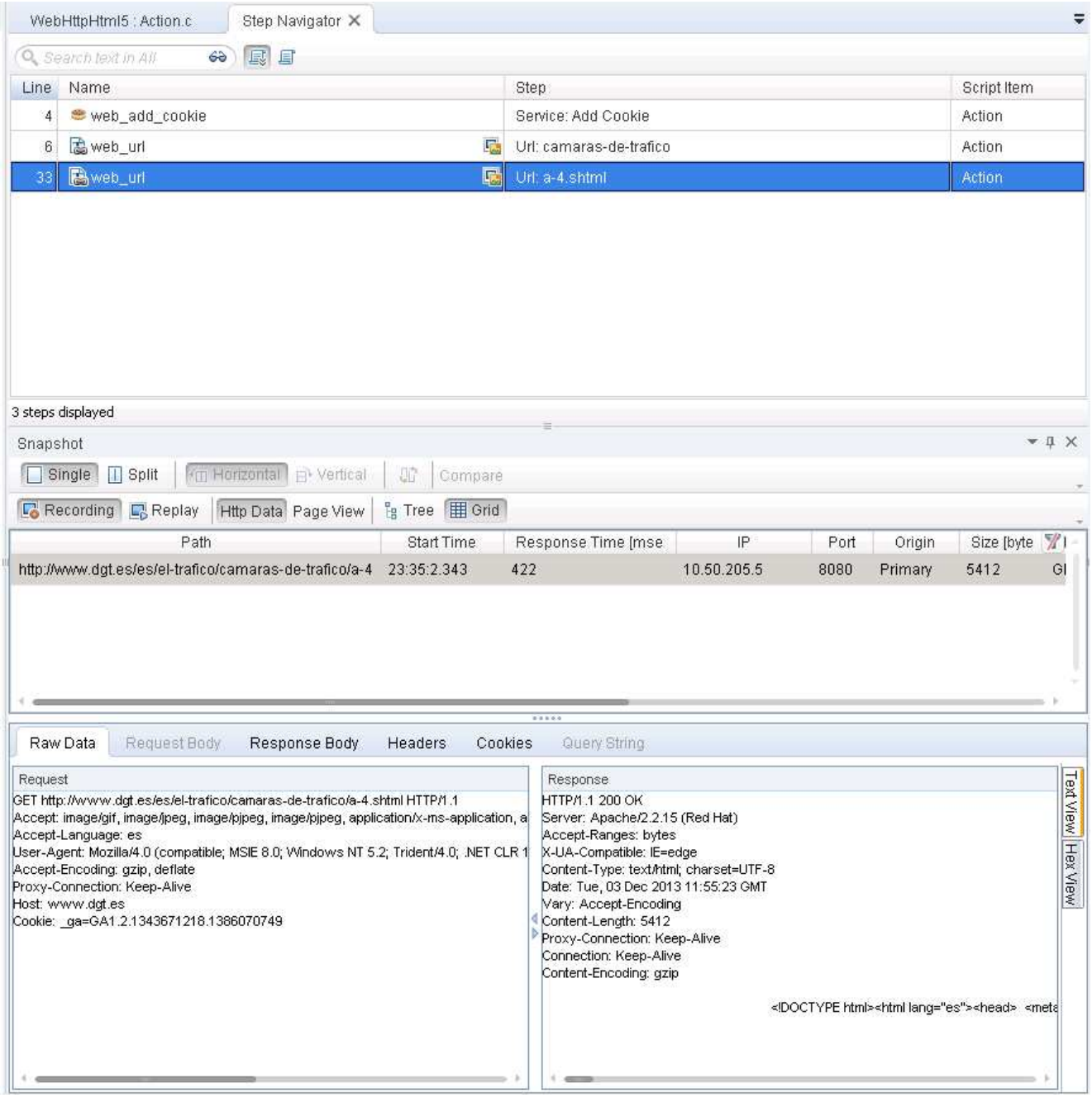


Ilustración 43. Detalle snapshot

5. En el menú, hacer click en "**View > Steps Toolbox**". La caja de herramientas de Steps aparecerá.
6. En la caja de herramientas Steps, en el campo de búsqueda, escribir web_reg, y seleccionar el paso **web_reg_find** en **Filter Results**.

7. Pinchar el icono de la flecha (**Add step to current location**). Aparece la siguiente pantalla, en la que estableceremos las opciones de búsqueda del String dentro de la Página Web ("Comienza por…", "termina por…", "mayúsculas", "minúsculas", etc…)

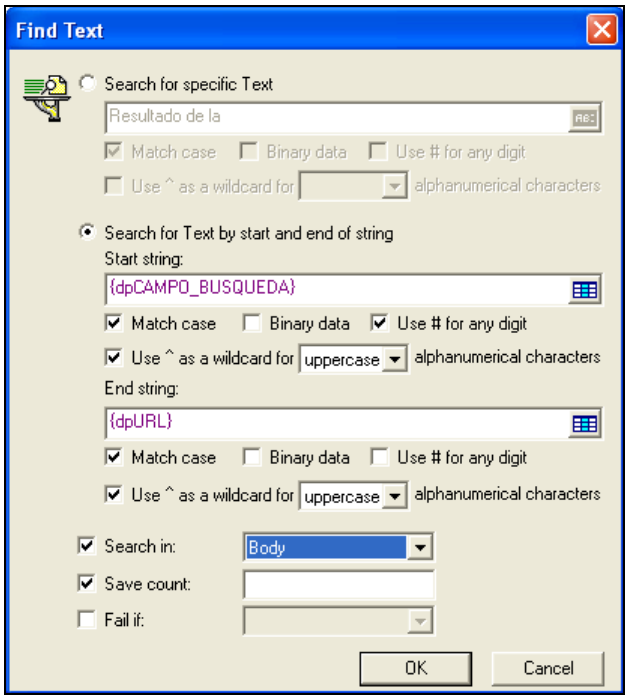


Ilustración 44. Ventana opciones de chequeo

8. Para más información, consultar la Ayuda y los Manuales de LoadRunner VuGen.

3.2.2.2.2.2 Método 2: Desde el Código Fuente del script:

Situarse en la llamada a la función **web_reg_find()** dentro del Código Fuente y utilizar las opciones, según se indica en la Ayuda y los Manuales de de **LoadRunner VuGen**.

3.2.2.2.2.3 Ejemplo

```
web_reg_find("Search=Body",
            "TextPfx/ALNUMUC={dpCAMPO_BUSQUEDA}",
            "TextSfx/DIG/ALNUMUC={dpURL}",
            LAST);
```

3.2.2.3 Caso: Comprobar si una búsqueda ha tenido éxito

3.2.2.3.1 Descripción

La función **web_reg_find()**, que permite buscar un String dentro del Código Fuente de una Página Web, admite opciones avanzadas de búsqueda.

La función **web_reg_save_param_ex()** se utilizar para realizar la **Correlación** y es la que nos permite guardar automáticamente los datos dinámicos generados y usarlos en las sucesivas navegaciones

3.2.2.3.2 Solución

Para saber si se ha tenido éxito en la búsqueda se puede utilizar una de las siguientes opciones:

a) **web_reg_save_param_ex("parameter_name","LB=XXX", "RB=YYY","Notfound=error", "ORD=1", LAST);**

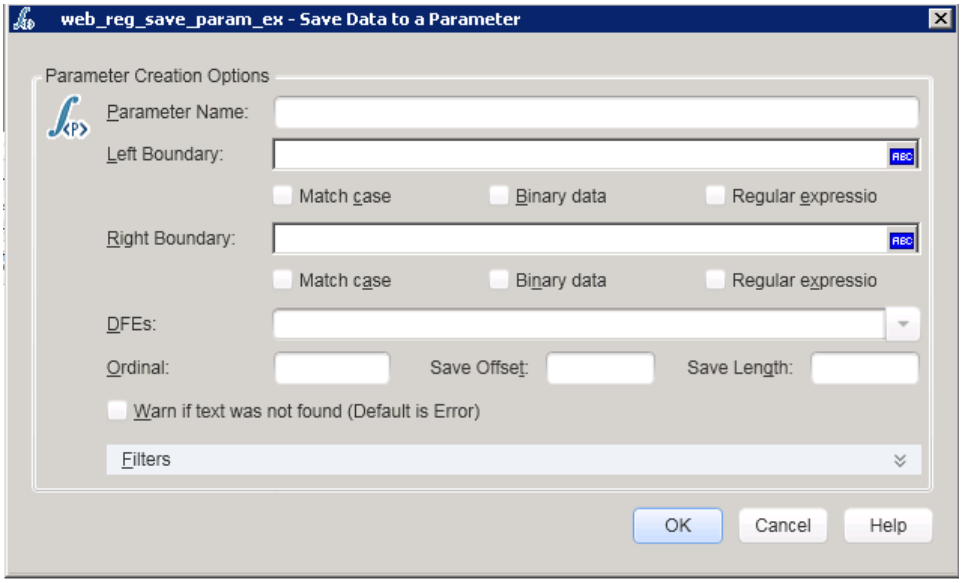


Ilustración 45. Ventana opciones correlación

En este caso se buscará en la respuesta del servidor los delimitadores "**XXX**" e "**YYY**", y se almacenará en el parámetro "**parameter_name**" lo que nos devuelva el servidor entre ellos. En caso de no encontrar los delimitadores, la función devolverá un error.

Una vez que hemos recuperado el dato devuelto por el servidor, utilizaremos la función **lr_eval_string("{parameter_name}")** para ver el contenido de dicho parámetro, y podremos evaluar si la búsqueda ha tenido éxito o no comparándolo con la respuesta esperada.

b) web_reg_find("Text=XXX", "SaveCount=parameter_name", LAST);

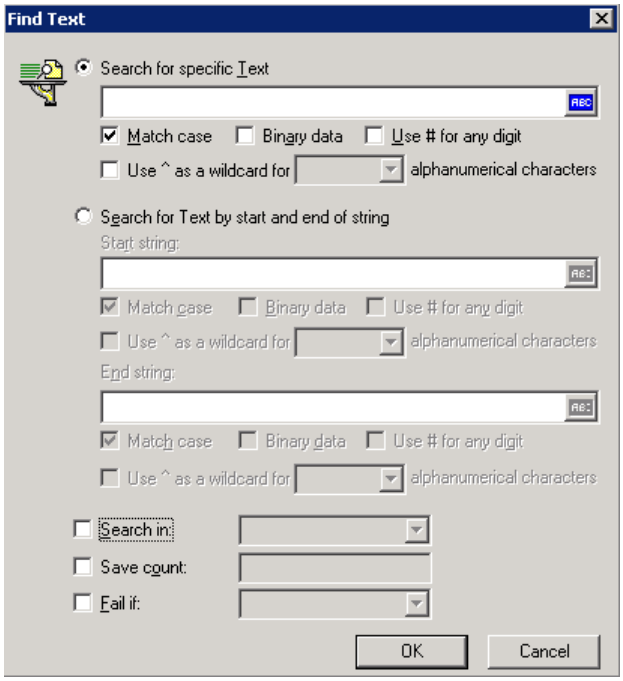


Ilustración 46. Ventana configuración web_reg_find()

En este caso se buscará en la respuesta del servidor el texto "**XXX**", y se almacenará en el parámetro "**parameter_name**" el número de ocurrencias de ese texto en la respuesta. En caso de no encontrar el texto, la función devolverá un "**0**".

Una vez que hemos recuperado el dato devuelto por el servidor, utilizaremos la función **lr_eval_string("{parameter_name}")** para ver el contenido de dicho parámetro, y podremos evaluar si la búsqueda ha tenido éxito o no comprobando si el valor obtenido es mayor que "**0**".

<NOTA>: lr_eval_string devuelve un char* con el valor actual del parámetro.

Otra opción es activar en el menú "**Vuser Run-time Settings/Log"** las siguientes opciones:

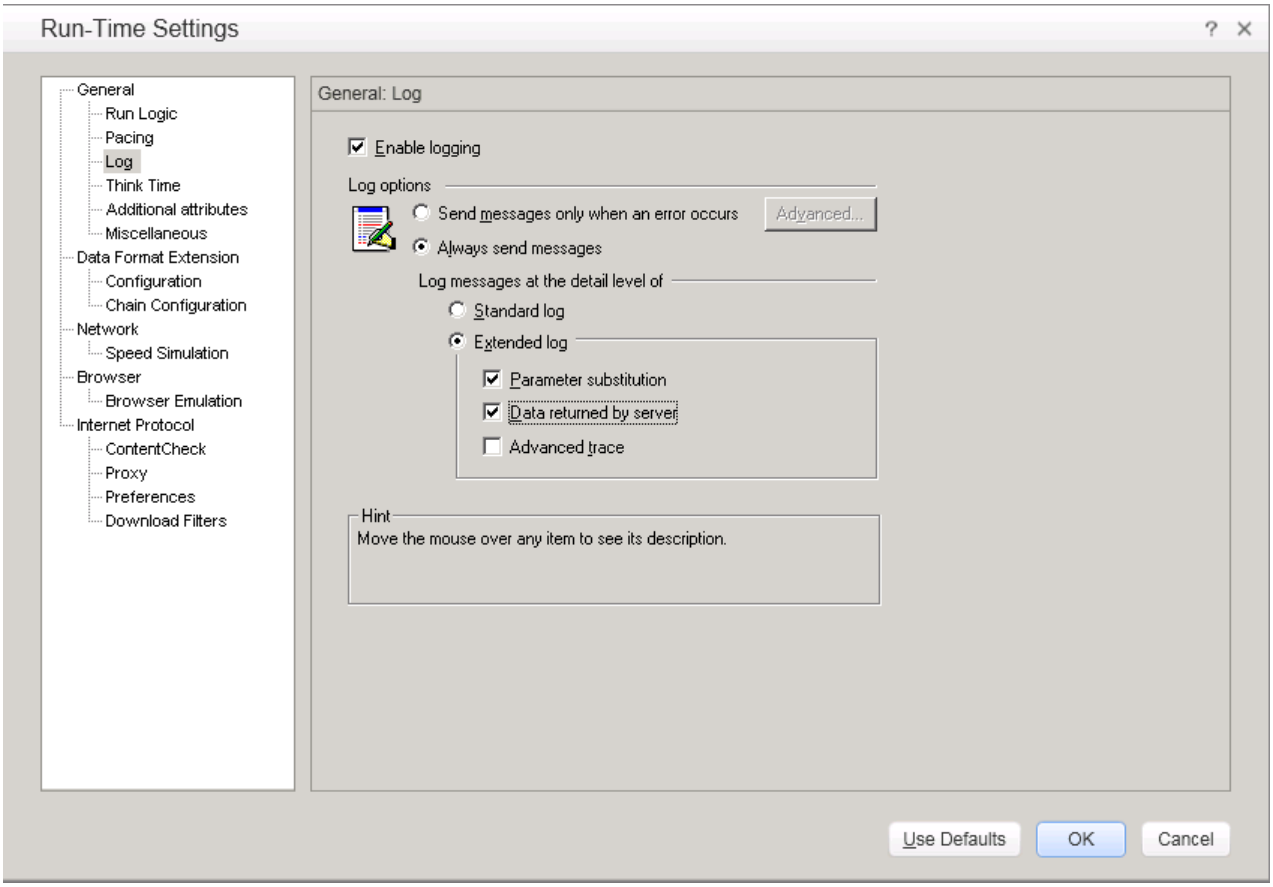


Ilustración 47. Ventana Run-Time Settings

Realizar una ejecución y ver en el **Replay Log** (Menu **View → Output)** si se han capturado/sustituido correctamente los parámetros.

3.2.2.4 Caso: Ejecutar un Formulario Web con un Body dinámico

3.2.2.4.1 Descripción

Puede darse el caso de que tengamos que invocar a un Formulario Web que tiene un número variable de parámetros, es decir, un número variable de campos a enviar al Servidor.

Por ejemplo, en una Aplicación tenemos un Formulario Web que nos permite ejecutar una consulta de Expedientes por un determinado criterio. Cuando rellenamos los campos por los cuales queremos realizar la consulta y hacemos click en el botón "**Aceptar**" (acción *submit* del Formulario Web de criterios de búsqueda), se genera un listado con los Expedientes que cumplen la condición.

Una vez que se ha generado el listado, podemos procesar a la vez todos los Expedientes devueltos por la consulta, seleccionándolos y pulsando click en el botón "**Aceptar**" (acción *submit* del listado que se ha generado tras la consulta).

Dependiendo del criterio de búsqueda que hayamos seleccionado, en algunos casos el listado que se genera devolverá un número de Expedientes y en otros casos, otro.El *Body* del listado se genera **dinámicamente**.

Si grabamos esta Transacción de negocio con **LoadRunner**, el número de Expedientes que se insertan en el script cuando se genera el listado de Expedientes **es estático**. Es decir, se insertan los Expedientes que **en ese momento** nos ha devuelto la consulta que hemos realizado.

El problema es que la función que LoadRunner inserta automáticamente para invocar al formulario, **web_submit_data(), no nos permite parametrizar el *Body***, dicho de otra forma, **no podemos parametrizar** el número de Expedientes que se han generado en el listado y que en consecuencia se envían al Servidor.

3.2.2.4.2 Solución: Utilizar la Función web_custom_request()

La función **web_custom_request()** permite parametrizar el *Body* de un Formulario Web, pudiendo "preparar" dinámicamente un *String* que contenga los elementos del Formulario Web y posteriormente enviarlo al Servidor (acción *submit*).

3.2.2.4.2.1 Sintaxis

```
int web_custom_request (const char *RequestName, <List of Attributes>,
[EXTRARES, <List of Resource Attributes>,] LAST );
```

Donde:

`RequestName`	**Es una etiqueta identificativa**
`<List of Attributes>`	**Lista de Atributos del Formulario Web**
`[EXTRARES, <List of Resource Attributes>,]`	**Lista de Atributos adicionales del Formulario Web**
`LAST`	**Indica el final de los argumentos**

3.2.2.4.2.2 Ejemplo

```
char* prmAuxBody;
char* prmHeader;
char* prmItems;
char* prmFoot;
…
/* Crear dinámicamente el Body del Formulario Web (Variable de tipo C: prmAuxBody) */

strcpy(prmAuxBody, prmHeader);
strcat(prmAuxBody, prmItems);
strcat(prmAuxBody, prmFoot);

/* Volcar el contenido de la variable que contiene el Body (Variable de tipo C: prmAuxBody) en un Parámetro
de tipo LoadRunner (Variable prmBody) */


lr_save_string (prmAuxBody, "prmBody");

/* Ejecutar el Formulario Web, cuyo Body es dinámico */
web_custom_request("guardarImportesCAAP.do",
                   "URL={dpUrl}/guardarImportesCAAP.do",
                   "Method=POST",
                   "Resource=0",
                   "RecContentType=text/html",
                   "Referer={dpUrl}/loadImportesCAAP.do",
                   "Snapshot=t20.inf",
                   "Mode=HTML",
                   "Body={prmBody}",
                   LAST);
```

3.2.2.5 Caso: Realizar búsquedas condicionales en llamadas a la función web_reg_find()

3.2.2.5.1 Descripción

En ocasiones la llamada a la función **web_reg_find()** se debe realizar no sobre un único patrón de búsqueda sino sobre varios, ya que el servidor devuelve diferentes textos, dependiendo de las opciones o parámetros que se ejecutan en el script.

Por ejemplo, en una transacción de altas de personal, pudiera darse el caso, que en función del sexo del sujeto dado de alta, el texto del mensaje de confirmación fuese distinto. Fem: “Actualizada Paloma”, Masc: “Actualizado Ramón”

3.2.2.5.2 Solución

En estos casos, la operativa a seguir se dividiría en varios apartados:

a) Utilizar la función **weg_reg_find()** tantas veces como textos a buscar. En cada una de las funciones utilizadas, almacenar en una variable el número de coincidencias encontradas.

b) Realizar la petición al servidor, que nos devolverá la página donde buscar los textos que hemos introducido en el apartado a).

c) Implementar las comparaciones, utilizando las variables almacenadas, que den como resultado el éxito o error de las búsquedas realizadas.

3.2.2.5.2.1 Ejemplo

En este ejemplo se pretende buscar el texto “Actualizado” o “Actualizada”, que es el que confirma el éxito de la transacción.

a)
```
web_reg_find("Text=Actualizado", “SaveCount=prmMasculino", LAST);
web_reg_find("Text=Actualizada", "SaveCount=prmFemenino", LAST);
```

b)
```
web_url("altaPersonal.php", "URL=altaPersonal.php?dni=03534556",
        "Resource=0", "RecContentType=text/html", "Referer=menu.html",
        "Snapshot=t2.inf", "Mode=HTML", LAST);
```

c)
```
if ( atoi(lr_eval_string("{prmMasculino}"))>0 ||  atoi(lr_eval_string("{prmFemenino}")) > 0)
{
        lr_output_message("Alta realizada con Éxito.");
  }
else
  {
        lr_fail_trans_with_error ("Error de Página");
        return(0);
}
```

3.2.2.6 Caso: Grabación de login mediante DataPower

3.2.2.6.1 Descripción

El proceso de login mediante DataPower se puede grabar sin problemas con la herramienta Vugen, pero en ocasiones la ejecución de dicho proceso presenta el siguiente error en el Replay Log de la herramienta:

“Error -32998: Invalid host name in URL="https:///""

Este error da como consecuencia que la petición de login falle también.

El problema se produce porque LoadRunner no gestiona una de las cookies que devuelve el servidor de aplicaciones.

3.2.2.6.2 Solución

Para solucionar el problema se debe introducir la cookie manualmente en el script de LoadRunner. Los pasos a seguir son los siguientes:

a) En el log Generation Log del Vugen se debe buscar la cookie “**WASReqURL**”. Aparecerá un texto del siguiente tipo:

Set-Cookie: WASReqURL=http://:443/WEB_TTRA_PORTAL/; Path=/WEB_TTRA_PORTAL

b) Añadir la cookie en el script de LoadRunner justo delante de la petición que realiza el login. La función de LoadRunner para añadir una cookie es **web_add_cookie**(). Esta función tiene los siguientes parámetros:

- **Name**=valor cookie (obligatorio). Este valor se obtiene del log Generation Log del Vugen.
- **Domain**=nombre del dominio (obligatorio). Este valor es el servidor.
- **Path**=por defecto es "/".

Quedaría de la siguiente manera:

```
web_add_cookie("WASReqURL=http://:443/WEB_TTRA_PORTAL;DOMAIN=intranet.pre-
datapower.trafico.es; path=/");
```

3.2.2.6.3 Ejemplo

A continuación se muestra un ejemplo de cómo quedaría el script después de introducir la cookie:

```
web_add_cookie("WASReqURL=http://:443/WEB_TTRA_PORTAL; DOMAIN=intranet.pre-
datapower.trafico.es; path=/");

web_submit_data("j_security_check",
        "Action=https://intranet.pre-datapower.trafico.es/WEB_TTRA_PORTAL/j_security_check",
        "Method=POST",
        "RecContentType=text/html",
        "Referer=https://intranet.pre-datapower.trafico.es/WEB_TTRA_PORTAL/",
        "Snapshot=t9.inf",
        "Mode=HTML",
        ITEMDATA,
        "Name=j_username", "Value=999977180", ENDITEM,
        "Name=j_password", "Value=TTRA180", ENDITEM,
        EXTRARES,
        "Url=../WEB_TTRA_PORTAL/images/cab1.jpg", "Referer=https://{dpURL}/WEB_TTRA/", ENDITEM,
        "Url=../WEB_TTRA_PORTAL/images/relleno.gif", "Referer=https://{dpURL}/WEB_TTRA/", ENDITEM,
        "Url=../WEB_TTRA_PORTAL/images/icon.png", "Referer=https://{dpURL}/WEB_TTRA/", ENDITEM,
        "Url=../WEB_TTRA_PORTAL/images/ico_.jpg", "Referer=https://{dpURL}/WEB_TTRA/", ENDITEM,
        "Url=../WEB_TTRA_PORTAL/images/flechaUrl.gif", "Referer=https://{dpURL}/WEB_TTRA/", ENDITEM,
```

"Url=../WEB_TTRA_PORTAL/images/relleno.jpg", "Referer=https://{dpURL}/WEB_TTRA/", ENDITEM, LAST);

3.2.2.7 Caso: Grabación de login mediante DataPower II

3.2.2.7.1 Descripción

Por lo general, durante las automatizaciones de scripts Vugen con DataPower, no se graba la pantalla de login, al buscar la pantalla, no aparece en el log.

3.2.2.7.2 Solución

Para solucionar el problema se debe introducir manualmente en el script de LoadRunner el siguiente código, antes de las primeras líneas grabadas por el script:

lr_continue_on_error(1);
web_reg_find("Text=**Introduzca sus credenciales**",**LAST**);
web_url("check.faces",
"URL=**https://{url acceso a la aplicación}** ",
"Resource=**0**",
"RecContentType=**text/html**",
"Referer=",
"Snapshot=**t1.inf**",
"Mode=**HTTP**",
LAST);
lr_continue_on_error(0);

Nota: al introducir manualmente estas líneas, la ejecución del script devolverá un hhtp 401, que es controlado mediante la función lr_continue_on_error(), que permite al script continuar la ejecución del script a pesar del error http.

3.2.2.7.3 Ejemplo

A continuación se muestra un ejemplo de cómo quedaría el script después de introducir este bloque:

```
vuser_init()
{
	web_set_sockets_option ("SSL_VERSION", "TLS");

	lr_start_transaction("TRAN_CADR_000_Pagina_Inicial");

	lr_continue_on_error(1);
	web_reg_find("Text=Introduzca sus credenciales",LAST);
	web_url("check.faces",
		"URL=https://pr-aplicaciones-pre.trafico.es/WEB_CADR/INIT/check.faces",
		"Resource=0",
		"RecContentType=text/html",
		"Referer=",
		"Snapshot=t1.inf",
		"Mode=HTTP",
		LAST);
	lr_continue_on_error(0);
	…
	…
	lr_end_transaction("TRAN_CADR_000_Cerrar_Sesion",LR_AUTO);
```

3.2.2.8 Caso: Grabación y ejecución de scripts con certificado digital sin WinInet.

3.2.2.8.1 Descripción

Al grabar un script de un proceso de negocio que utiliza certificado digital se producen problemas de grabación o al grabarlo con WinInet se graba correctamente pero no permite la ejecución sin la opción "**WinInet replay instead of Sockets**" desactivada.

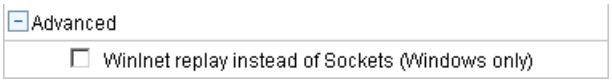


Ilustración 48. Opción WinInet replay instead of Sockets (Windows only)

Habitualmente, cuando se produce este problema, el certificado se incluye en el script de Loadrunner con este formato:

```
web_set_certificate_ex(
  "CertIndex=1",
  LAST);
```

Este problema se produce porque al importar el certificado digital en el navegador que se usa para grabar el script no se ha activado la opción de permitir exportarlo.

3.2.2.8.2 Solución

Para solucionar el problema se debe seguir la siguiente operativa:

1.- En el Internet Explorer, eliminar todos los certificados actuales.
2.- Importar de nuevo el certificado .pfx (o .p12). En las opciones dejar desactivado "Habilitar protección segura de claves privadas" y activado "Marcar esta clave como exportable"
3.- Volver a grabar el script, con las opciones de grabación **Capture level** con valor **WinINet level data**.
4.- Comprobar que en el script se ha generado un código de este tipo:

```
web_set_certificate_ex("CertFilePath=WinINetCert1.pem",
```

```
"CertFormat=PEM",
"KeyFilePath=WinINetCert1.pem",
"KeyFormat=PEM",
"Password=51a5dff9",
"CertIndex=1",
LAST);
```

5.- Verificar que el archivo **WinINetCert1.pem** que se genera en la carpeta del script tiene tamaño mayor que 0.

6.- Verificar que en las opciones de ejecución se encuentra **desmarcada** la casilla "**WinInet replay instead of Sockets**"

3.2.3 Protocolo: WebServices

3.2.3.1 Caso: Crear un Script sencillo de Protocolo WebServices con una llamada a la Función soap_request()

3.2.3.1.1 Descripción

Si el WebService que tenemos que ejecutar no tienen parámetros complejos, como por ejemplo, campos que sean de tipo XML, podemos utilizar directamente una llamada a la función **soap_request()**, pasándole por parámetro el mensaje SOAP (SOAP Envelope).

3.2.3.1.2 Solución: Utilizar la Función soap_request()

En esta función, el parámetro en el que se envía el mensaje SOAP (SOAP Envelope) es un String, de tal manera que podemos construirlo en tiempo de ejecución, copiar su contenido de la llamada que se hace desde otro Sistema y pegarlo, posteriormente parametrizarlo por partes a nivel de parámetros de **LoadRunner**, etc…

El formato del mensaje SOAP a enviar es el siguiente:

SOAPEnvelope=<SOAP-ENV:Envelope xmlns:SOAP-ENV=
…
</SOAP-ENV:Envelope>

<NOTA>: La persona vinculada a la Aplicación a probar debe proporcionarnos este mensaje SOAP y/o la información necesaria para construirlo

3.2.3.1.2.1 Sintaxis

```
int soap_request (const char *StepName, URL, <XMLEnvelope>, LAST);
```

Donde:

`Stepname`	**Es una etiqueta identificativa**
`URL`	**Es el *endpoint* del WebService**
`<XMLEnvelope>`	**Es el mensaje SOAP en formato:** `SOAPEnvelope=<SOAP-ENV:Envelope xmlns:SOAP-ENV= … </SOAP-ENV:Envelope>`
`LAST`	**Indica el final de los argumentos**

3.2.3.1.2.2 Ejemplo

1. Ejecutar **LoadRunner VuGen.**
2. Ejecutar la opción: "**File -> New Script and Solution**".
3. En la pantalla que aparece a continuación, seleccionar el Protocolo "**WebServices**" y pulsar el botón "**Create**".

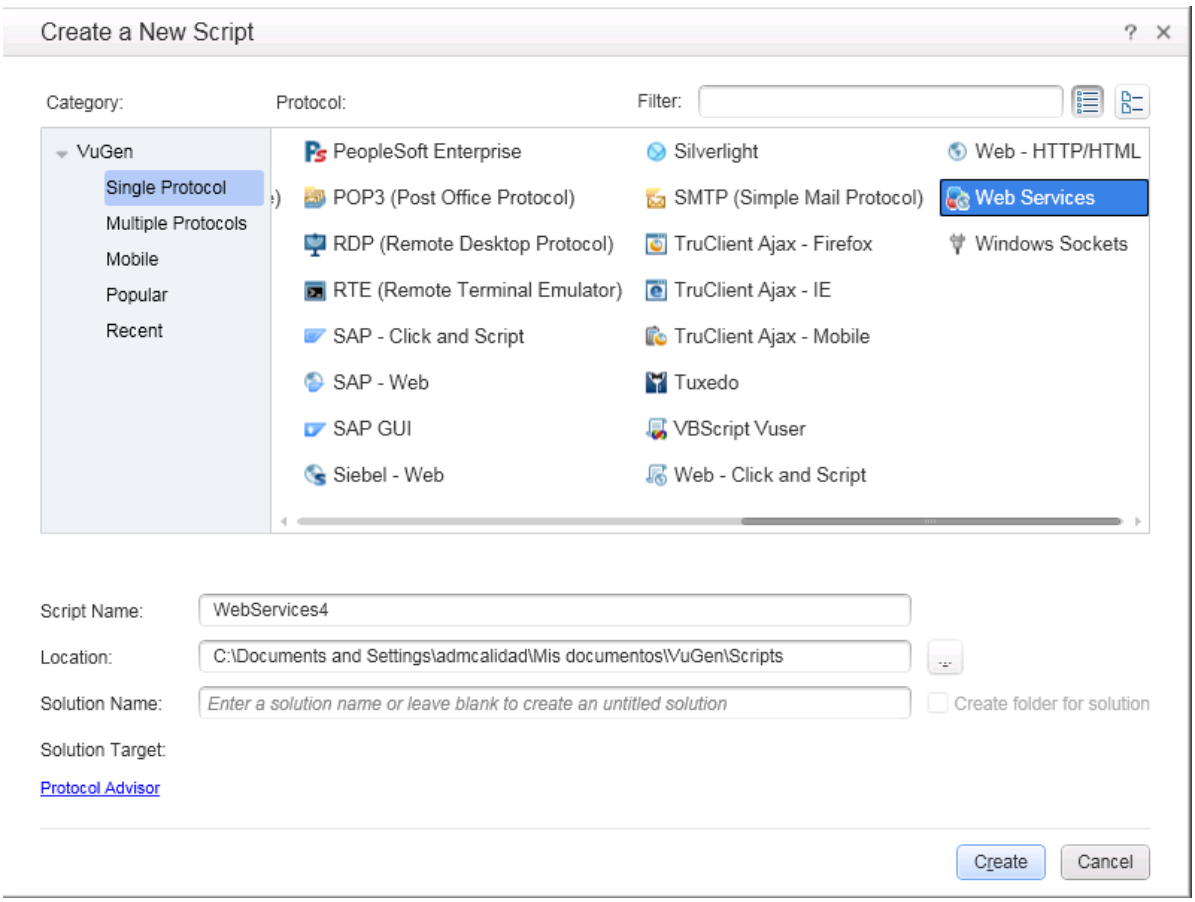


Ilustración 49. Ventana Create a New Script

4. Seleccionar la función **Action()** y situarse con el cursor antes de la sentencia "**return 0;**"

Ilustración 50. Panel Solution Explorer

5. Hacer click con el botón derecho del ratón y seleccionar la opción: "**Insert -> New Step…**"

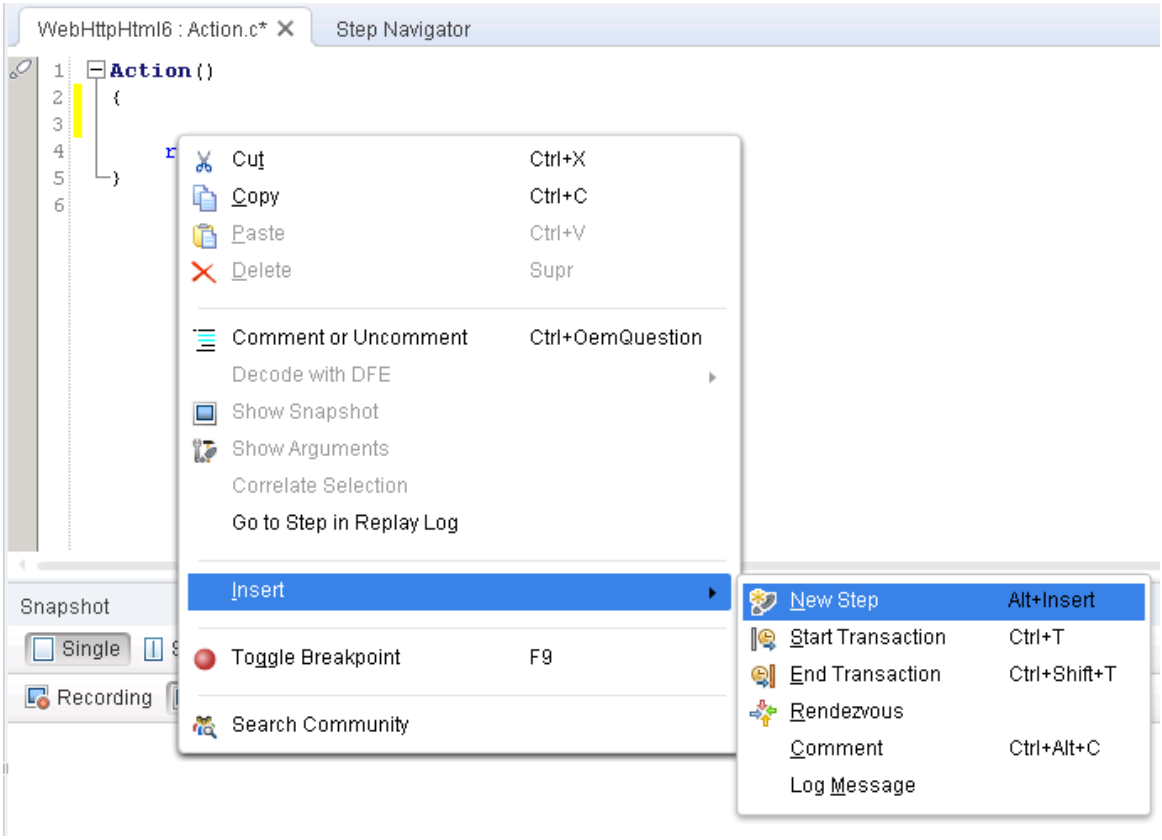


Ilustración 51. Opción para insertar un nuevo paso

6. En la caja de herramientas de Steps, en el campo de búsqueda escribir "soap" , y seleccionar la opción: "**Soap_request**" y pulsar el icono de la flecha (**Add step to current location**).

7. En la pantalla que aparece a continuación, tenemos que introducir la información relativa a la llamada que vamos a hacer a la función **soap_request(),** es decir, la llamada al WebService:

StepName:	**Step_000**
URL:	La URL del WebService al que vamos a llamar
SOAPEnvelope:	El mensaje SOAP en formato XML Podemos copiar y pegar el contenido desde el fichero XML o directamente importar el fichero XML

<NOTA>: La persona vinculada a la Aplicación a probar debe proporcionarnos este mensaje SOAP y/o la información necesaria para construirlo

SOAPAction: *Dejar el Campo vacío*

Snapshot: **t001.inf**

ResponseParam: Nombre de una variable (de tipo parámetro de **LoadRunner**) en la que se va a almacenar el *return* de la llamada al WebService, es decir, un String XML. Después de la llamada, podemos buscar un String dentro de este parámetro para ver si se ha ejecutado correctamente; por ejemplo si devuelve el contenido de un determinado un Campo en concreto.
Para esto podemos utilizar la función de C: **strstr()**

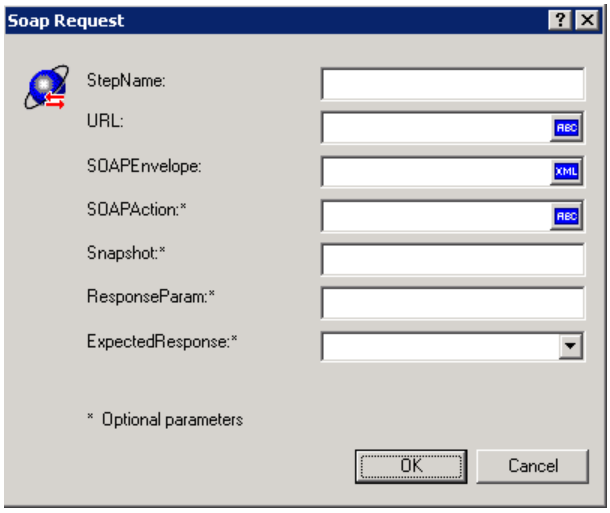


Ilustración 52. Ventana Soap Request

8. Pulsar el botón "**OK**".
9. Como podemos comprobar, en el código fuente del script se ha introducido una llamada a la función **soap_request()** con los parámetros que hemos indicado en el asistente:

```
Action()
{

    soap_request("StepName=Step_000",
        "URL=http://datapower.trafico.es:7000/Consulta",
        "SOAPEnvelope=<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
        "\r\n<soapenv:Envelope\r\n\txmlns:wsse="
        "\"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0."
        "xsd\"\r\n\txmlns:soapenv="
        "\"http://schemas.xmlsoap.org/soap/envelope/\"\r\n\txmlns:ant="
        "\"http://antecedentesvehiculos.webservices.trafico.es\"\t\r\n\txmlns:xs="
        "\"http://www.w3.org/2001/XMLSchema\">\r\n\t<soapenv:Header>"
        "\r\n\t\t<wsse:Security>\r\n\t\t\t<wsse:UsernameToken>\r\n\t\t\t\t<wsse:Username>"
        "12345678</wsse:Username>\r\n\t\t\t\t<wsse:Password>DGT678</wsse:Password>"
        "\r\n\t\t\t</wsse:UsernameToken>\r\n\t\t</wsse:Security>\r\n\t</soapenv:Header>"
        "\r\n\t<soapenv:Body>\r\n\t<ant:consultarVehiculo>\r\n\t\t<ant:matricula>"
        "CS2222PP</ant:matricula>\r\n\t</ant:consultarVehiculo>\r\n\t</soapenv:Body>"
        "\r\n</soapenv:Envelope>",
        "Snapshot=t001.inf",
        "ResponseParam=szResult",
        LAST);

    return 0;
}
```

Ilustración 53. Ejemplo petición soap_request

10. Situarse con el cursor antes de la sentencia **soap_request().**
11. Añadir la siguiente sentencia:

```
web_add_header("SOAPAction", "\"urn:SOAPAction\"");
```

12. Ejecutar el script y comprobar el resultado devuelto (nos aseguramos que el contenido de la variable que almacena el return de la llamada es el correcto, como se puede ver en **3.1.5 Chequeos de contenido con WebServices**).

3.2.3.2 Caso: Ejecutar un WebService con un mensaje SOAP dinámico (parametrizar un XML)

3.2.3.2.1 Descripción

En **LoadRunner**, por defecto, cuando se graba un script de protocolo "**Web Services**" y se llama a un Servicio Web, la función que se inserta automáticamente es: **web_service_call()**

Esta función admite como parámetro el mensaje SOAP. El problema es que si el mensaje SOAP que queremos enviar tiene campos que son de tipo XML, la llamada falla porque no se encuentra el delimitador de fin de mensaje.

3.2.3.2.2 Solución 1: Utilizar la Función soap_request()

En esta función, el parámetro en el que se envía el mensaje SOAP (SOAP Envelope) es un String, de tal manera que podemos construirlo en tiempo de ejecución, copiar su contenido de la llamada que se hace desde otro Sistema y pegarlo, parametrizarlo por partes a nivel de parámetros de **LoadRunner**, etc…

El formato del mensaje SOAP a enviar es el siguiente:

```
SOAPEnvelope=<SOAP-ENV:Envelope xmlns:SOAP-ENV=
…
</SOAP-ENV:Envelope>
```

<NOTA>: La persona vinculada a la Aplicación a probar debe proporcionarnos este mensaje SOAP y/o la información necesaria para construirlo

3.2.3.2.2.1 Sintaxis

```
int soap_request (const char *StepName, URL, <XMLEnvelope>, LAST);
```

Donde:

`Stepname`	**Es una etiqueta identificativa**
`URL`	**Es el *endpoint* del WebService**
`<XMLEnvelope>`	**Es el mensaje SOAP en formato:** `SOAPEnvelope=<SOAP-ENV:Envelope xmlns:SOAP-ENV= … </SOAP-ENV:Envelope>`
`LAST`	**Indica el final de los argumentos**

3.2.3.2.2.2 Ejemplo

> **<NOTA>:** Antes de llamar a la Función `soap_request()` tenemos que realizar una llamada a la Función `web_add_header()`, pasándole por parámetro el nombre de una etiqueta identificativa

```
web_add_header("SOAPAction", "\"urn:Action_registrarSalida\"")

soap_request("Stepname=registrarSalida_101",
"URL=http://{dpURL}/rt/services/RegistroTelematicoSalida",
"SOAPEnvelope= <SOAP-ENV:Envelope xmlns:SOAP-ENV='"
"http://schemas.xmlsoap.org/soap/envelope/' xmlns:q0='"
"http://ws.registry.rtn.ieci' xmlns:xsd='http://www.w3.org/2001/XMLSchema' "
"xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'><SOAP-ENV:Body>"
"<q0:registrarSalida><idSession>{prmConexionReturn}</idSession><infoReg>"
"<idTramite>{prmIdtramite}</idTramite><documentos></documentos><idioma>{prmIdioma}</idioma><xmlSo
licitud>{prmXmlsolicitud}</xmlSolicitud></infoReg></q0:registrarSalida></SOAP-ENV:Body></SOAP-
ENV:Envelope>",
"Snapshot=t1.inf",
"ResponseParam=prmSalidaReturn",
LAST);
```

Los parámetros indicados en **azul negrita** como por ejemplo: **{prmConexionReturn}**, son parámetros de **LoadRunner**, es decir, pueden ser parámetros que "tiren" de un fichero de datos de entrada. En el caso de que estos parámetros sean de tipo XML, tenemos que encerrarlos entre los siguientes delimitadores, para que así se anule la ambigüedad:

```
<![CDATA[
…
]]>
```

Por ejemplo:

<![CDATA**[<?xml version='1.0' encoding='UTF-8'?><Solicitud_Registro_Salida Versin='1.0'><Datos_Genericos><Organismo>DGT</Organismo></Datos_Genericos></Solicitud_Registro_Salida>**]]>

El contenido en **negrita** es el contenido del parámetro de **LoadRunner**.

A continuación se adjunta un ejemplo real de fichero de datos de entrada que contiene este tipo de parámetros:

Ilustración 54. Ventana para salvar el script

3.2.3.2.3 Solución 2: Introducir la llamada SOAP mediante el asistente.

3.2.3.2.3.1 Ejemplo

1. Desde el ***Vugen***, añadimos un nuevo servicio a través de un **WSDL**, seleccionando el icono "**Manage services**" o accediendo al menú "**SOA Tools → Manage Services**":

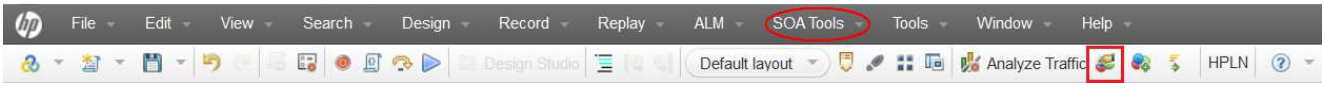


Ilustración 55. Ventana para salvar el script

2. Pulsamos "**Import**" en la ventana que aparece:

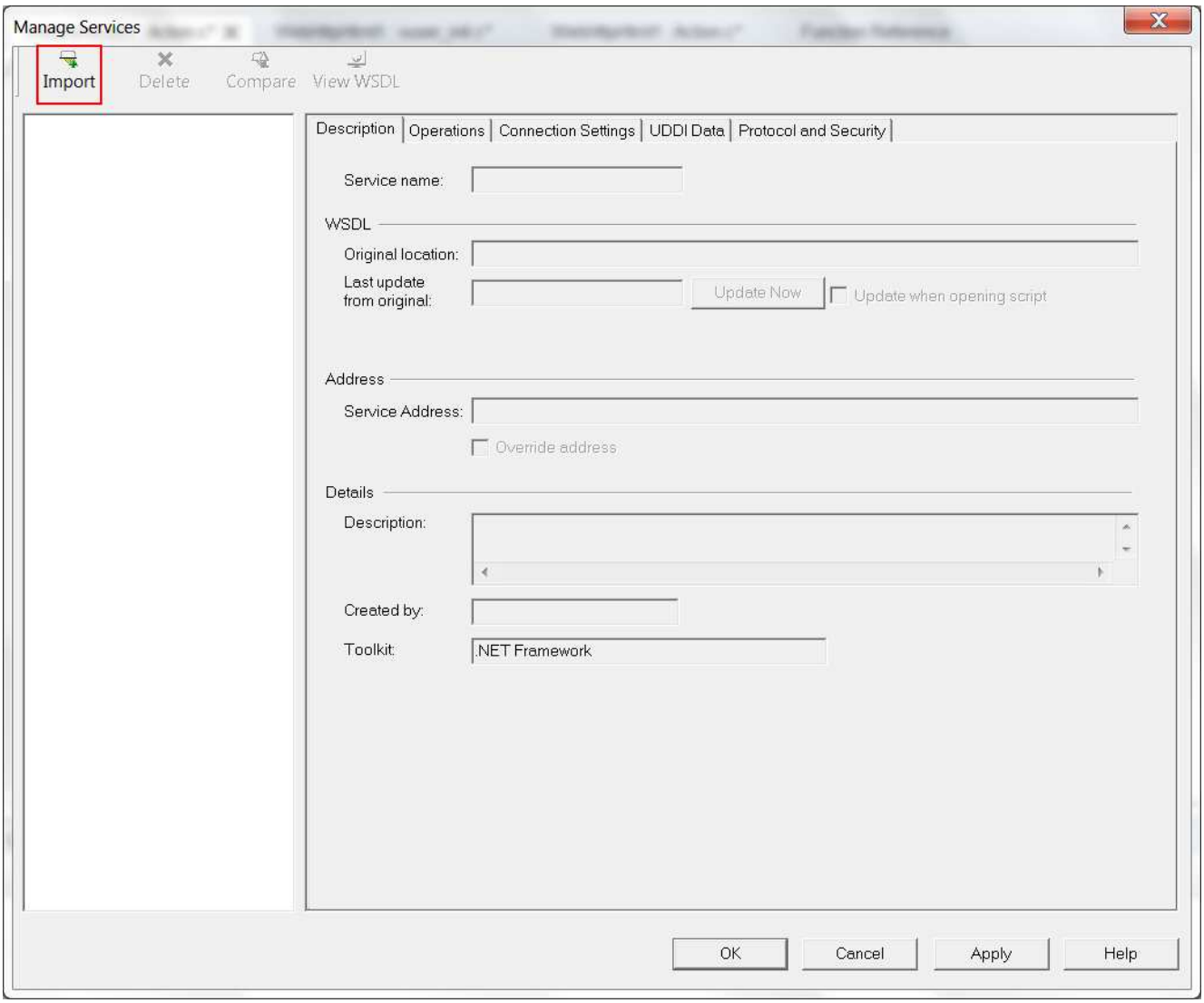


Ilustración 56. Ventana Manage Services

3. Seleccionamos si queremos importar el **WSDL** desde una **URL** o desde un **fichero**, escribimos la ruta correspondiente y pulsamos "**Import**":

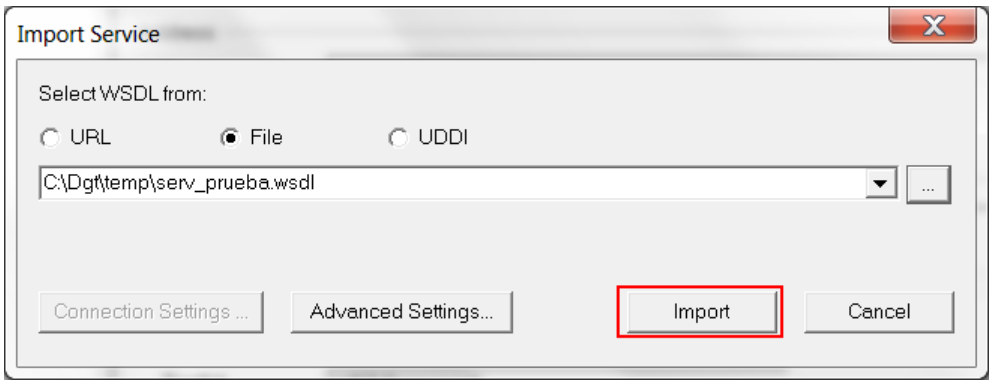


Ilustración 57. Ventana Import Service

4. Una vez importado el servicio, pulsamos en "**OK**":

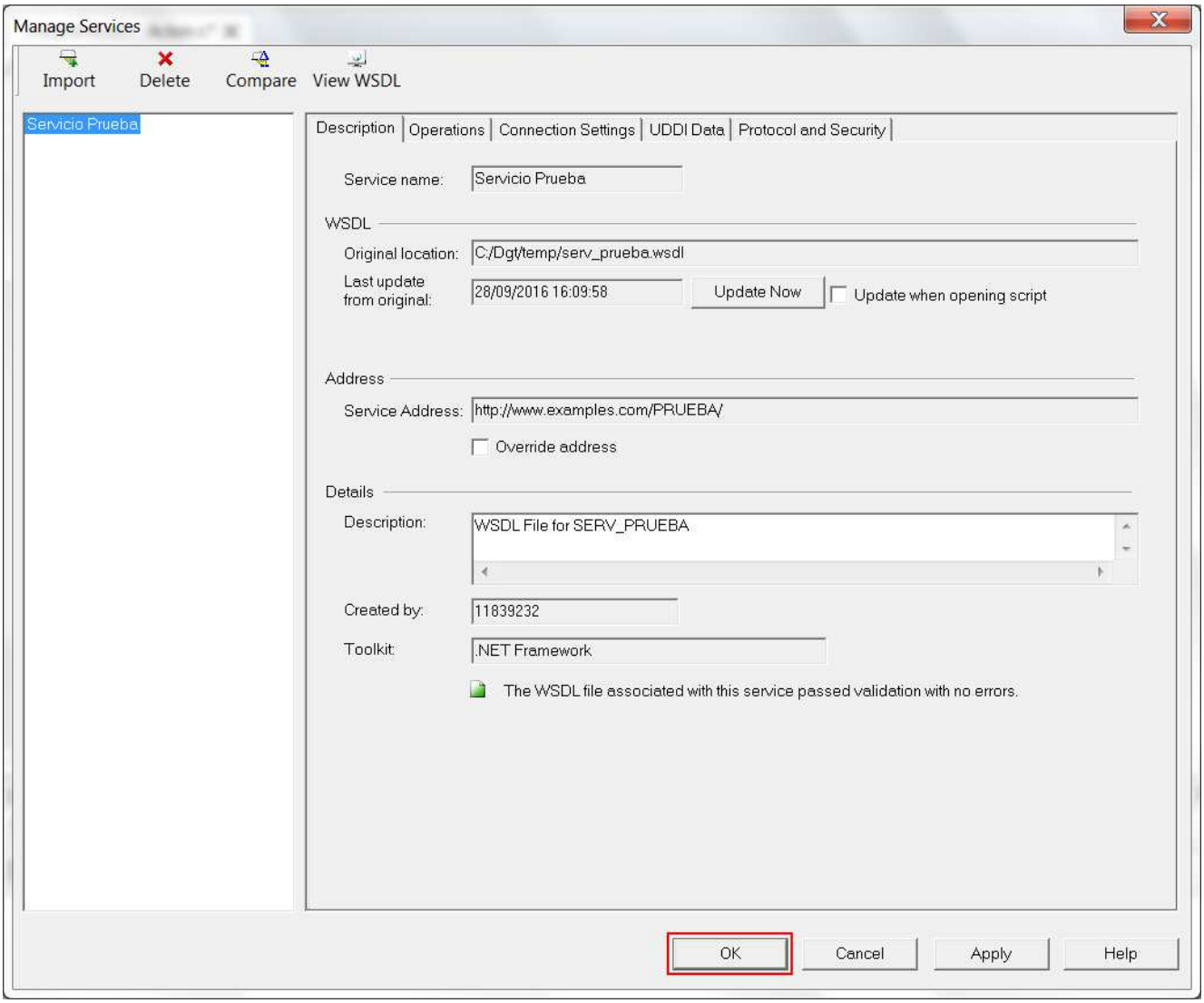


Ilustración 58. Importación de un servicio

5. Desde el ***Vugen***, añadimos un nuevo servicio, seleccionando el icono "**Add Service Call**" o accediendo al menú "**SOA Tools → Add Service Call**":

Ilustración 59. Opción SOA Tools

6. En la pantalla que nos muestra, se elige el servicio y operación y se selecciona el objeto donde se va a introducir la llamada SOAP que nos han facilitado.

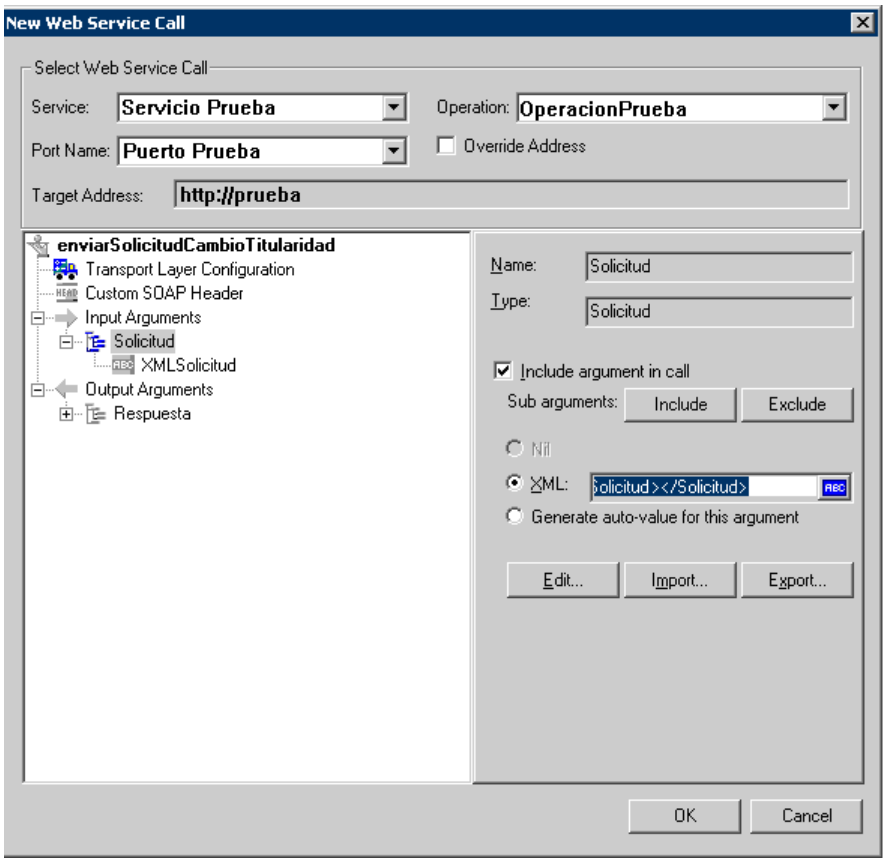


Ilustración 60. Ventana para seleccionar llamada a servicio

7. Pulsar el botón "**Edit**".
8. En la pantalla que aparece nos vamos a la Pestaña: "**Text View**" y pulsamos: "**Import File**"

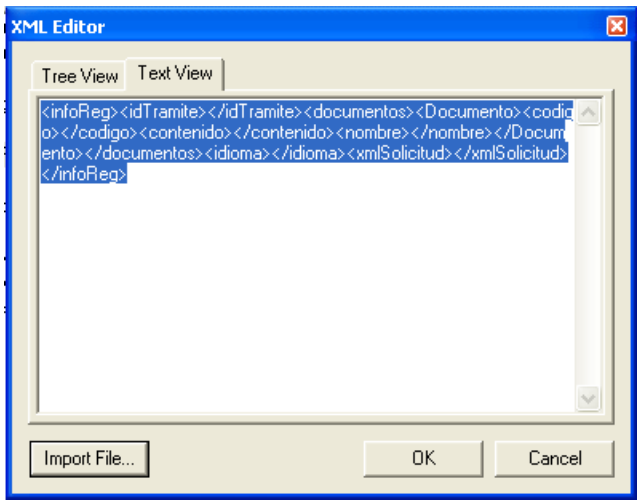


Ilustración 61. Editor xml

9. Seleccionamos el fichero XML que queremos importar y que nos cargará en la ventana "**XML Editor**". En dicha ventana eliminamos todo el código de las cabeceras SOAP hasta que nos quede sólo el código de la estructura del objeto. En este caso la estructura del objeto "**infoReg**".
10. Pulsamos "**OK**" y en la siguiente pantalla otra vez "**OK**". En la parte del código del script **VuGen**, nos aparecerá el código de la llamada SOAP que acabamos de insertar. Dicho contenido se podrá parametrizar.

```
lr_start_transaction("TRAN_004_Registrar_Salida_Sin_Doc");


  web_service_call( "StepName=registrarSalida_102",
    "SOAPMethod=RegistroTelematicoSalidaService.RegistroTelematicoSalida.registrarSalida",
    "ResponseParam=response",
    "Service=RegistroTelematicoSalidaService",
    "Snapshot=t1199873477.inf",
    BEGIN_ARGUMENTS,
    "idSession={Param_conexionReturn}",
    "xml:infoReg=<infoReg><idTramite>BAT</idTramite><documentos><Documento><codigo>"
        "</codigo><contenido></contenido><nombre></nombre></Documento></documentos>"
        "<idioma>es</idioma><xmlSolicitud>&lt;?xml version=\"1.0\" encoding=\"UTF-8\"?&"
        "gt;&lt;Solicitud_Registro_Salida version=\"1.0\"&gt;&lt;Datos_Genericos&gt;&lt;"
        "Organismo&gt;DGT&lt;/Organismo&gt;&lt;Numero_Registro_Entrada&gt;"
        "200712800000011&lt;/Numero_Registro_Entrada&gt;&lt;Destinatario&gt;&lt;Nombre&"
        "gt;Gonzalo&lt;/Nombre&gt;&lt;Apellidos&gt;Pérez Centeno&lt;/Apellidos&gt;&lt;"
        "Documento_Identificacion&gt;&lt;Tipo&gt;1&lt;/Tipo&gt;&lt;Numero&gt;11111111H&"
        "lt;/Numero&gt;&lt;/Documento_Identificacion&gt;&lt;Correo_Electronico&gt;"
        "pruebas@dgt.es&lt;/Correo_Electronico&gt;&lt;/Destinatario&gt;&lt;Interesados&"
        "gt;&lt;Interesado&gt;&lt;Nombre&gt;Gonzalo&lt;/Nombre&gt;&lt;Apellidos&gt;"
        "Vaquero Vaquero&lt;/Apellidos&gt;&lt;Documento_Identificacion&gt;&lt;Tipo&gt;1&"
        "lt;/Tipo&gt;&lt;Numero&gt;71134112B&lt;/Numero&gt;&lt;/Documento_Identificacion&"
        "gt;&lt;Correo_Electronico&gt;pruebas@dgt.es&lt;/Correo_Electronico&gt;&lt;"
        "/Interesado&gt;&lt;/Interesados&gt;&lt;Asunto&gt;&lt;Codigo&gt;BOC&lt;/Codigo&"
        "gt;&lt;Descripcion&gt;Solicitud de baja online para CATVs&lt;/Descripcion&gt;&"
        "lt;/Asunto&gt;&lt;Origen&gt;&lt;Codigo&gt;101001&lt;/Codigo&gt;&lt;Descripcion&"
        "gt;Dirección General de Tráfico&lt;/Descripcion&gt;&lt;/Origen&gt;&lt;"
        "Numero_Expediente/&gt;&lt;Matricula&gt;M 2772SW&lt;/Matricula&gt;&lt;Bastidor/&"
        "gt;&lt;/Datos_Genericos&gt;&lt;Datos_Especificos&gt;&lt;/Datos_Especificos&gt;&"
        "lt;Documentos/&gt;&lt;/Solicitud_Registro_Salida&gt;</xmlSolicitud></infoReg>",
    END_ARGUMENTS,
    BEGIN_RESULT,
        "registrarSalidaReturn=Param_registrarSalidaReturn",
    END_RESULT,
    LAST);
```

Ilustración 62. Contenido parametrizable

3.2.3.3 Caso: Crear un Script sencillo de Protocolo WebServices securizado mediante certificado digital.

3.2.3.3.1 Descripción

Si el WebService que tenemos que ejecutar requiere un certificado digital para autenticarse en el servidor.

3.2.3.3.2 Solución: Utilizar la Función web_set_certificate_ex()

En esta función, se carga un certificado en formato **pem** incluido en la carpeta del script para autenticarse correctamente en el servidor.

Este certificado debe obligatoriamente incluir la clave privada y estar en formato **pem**.

3.2.3.3.2.1 Sintaxis

```
web_set_certificate_ex(
     "CertFilePath=<certificado_destino>.pem",
     "CertFormat=PEM",
     "KeyFilePath=<certificado_destino>.pem ",
     "KeyFormat=PEM",
     "Password=<password del certificado>",
     LAST );
```

Donde:

`<certificado_destino>.pem`	**Es el certificado necesario.**
`<password del certificado>"`	**Es el *password* del certificado**

3.2.3.3.2.2 Ejemplo

1. <opcional> Exportar certificado desde el navegador. Menú “Opciones de Internet → Contenido → Certificados”, elegir el certificado deseado, botón “Exportar”, activar “Exportar la clave privada”, activar “Si es posible, incluir todos los certificados en la ruta de acceso de la aplicación” y desactivar las otras dos opciones, introducir contraseña y guardar el archivo.

2. Convertir el certificado con formato pfx en pem.

Desde una ventana ms-dos.

<carpeta de instalacion de VuGen>bin\openssl pkcs12 -in **<certificado_origen>.**pfx -out **<certificado_destino>.**pem

Enter Import Password: <**password del certificado>**
MAC verified OK
Enter PEM pass phrase: <**password del certificado>**

Verifying - Enter PEM pass phrase: **<password del certificado>**

Donde:

- **<carpeta de instalación de Loadrunner>** es la ruta donde se ha instalado VuGen, por defecto: C:\Archivos de programa\HP\Virtual User Generator.
- **<certificado origen>** Nombre del certificado origen en formato pfx.
- **<certificado destino>** Nombre deseado para el certificado PEM a generar.
- **<password del certificado>** Contraseña del certificado origen.

3. Copiar el certificado .pem a la carpeta donde esté el script a securizar. Añadir la llamada a **web_set_certificate_ex()** antes de la llamada al servicio web que se quiere securizar.