Desarrollo de aplicaciones multiplaforma en Linux

Aplicaciones en el escritorio con Java/SWT

Franco M. Catrin L.

Desarrollo de aplicaciones multiplaforma en Linux: Aplicaciones en el escritorio con Java/SWT

por Franco M. Catrin L.

Historial de revisiones

Revisión 1.0 13-10-2002 Revisado por: fcl

Propuesta inicial 3er encuentro Linux U.BioBio

Tabla de contenidos

Capítulo 1. Introducción

Linux ha logrado consolidar su posición en el campo de los servidores. Ya no hay dudas de que es una real alternativa como sistema operativo de reemplazo a Windows NT/2000 en incluso Unix'es. Sin embargo, en el area de los escritorios esto ha ido sucediendo de una forma más lenta.

En este documento se planteará una forma de enfrentar este problema en la realidad chilena.

Linux en el escitorio

Debemos distingir entre dos escenarios para Linux en el escritorio. Uno se encuentra en los hogares, en donde cada dia son mas los aficionados a la computación que han decidido instalar Linux en sus casas como una herramienta de aprendizaje y de uso cotidiano (web, mail, IM, multimedia, etc). Hasta el momento son pocos los usuarios no entendidos en computación que se estan iniciando en el uso de Linux con ayuda de los primeros, pero esa cifra va en aumento.

El segundo escenario es un poco mas complejo, y corresponde a los escritorios Linux usados en empresas. En este caso la factibilidad no solo depende de las habilidades del usuario, sino que de otros aspectos, como es el "entorno" de operación.

Estrategia para incorporar más escritorios Linux en la empresa

Para que se pueda ir migrando hacia Linux en el escritorio, se deben superar trabas que son de caracter operacional. Principalmente son:

- Conectividad con otros miembros de la red, a nivel de servicios
- Disponibilidad de aplicaciones nativas de uso general
- Disponibilidad de aplicaciones específicas del negocio

El primer punto está practicamente solucionado. Hay servidores/clientes disponibles para la gran mayoria de servicios standard. Y tambien hay servidores y clientes para servicios propietarios como SMB, usados intensamente en redes Windows.

El segundo punto tambien está solucionado o (en camino de) para las tareas tradicionales de oficina, que coinciden parcialmente con lo indicado inicialmente en un computador de escritorio hogareño.

El ultimo punto, la disponibilidad de aplicaciones específicas del negocio, es más dificil de solucionar. Esto se debe a que son aplicaciones desarrolladas a medida, y que por lo general se han venido arrastrando desde años. El esfuerzo de desarrollo invertido en terminos de costo e ingenieria es muy alto y es muy dificil reemplazarlas por aplicaciones nuevas de un dia para otro. Y añadido a esto, se tiene el hecho de que si se han realizado usando herramientas propietarias que producen código muy amarrado a la plataforma original (mayoritariamente Windows), es casi imposible lograr ejecutar estas aplicaciones si se realiza una migración a Linux.

Por un lado no se pueden pasar estos escritorios a Linux ya que no se pueden ejecutar las aplicaciones legacy que son dependientes de la plataforma. Y por otra parte no se puede iniciar el desarrollo de nuevas aplicaciones que se puedan ejecutar en Linux debido a que aun no se puede tener Linux instalado.

Si se tiene este escenario, existen dos caminos a seguir. Uno es buscar una forma de emular un entorno de ejecución legacy (dosemu, wine) o bien desarrollar aplicaciones multiplataforma que puedan ser ejecutadas en ambos entornos indistinatmenta, y con eso se podrá realizar la migración a Linux a medida que se terminen las dependencias de aplicaciones legacy.

Este trabajo intenta cubrir esta ultima alternativa: Desarrollo de aplicaciones multiplataforma para escritorio en Linux, de tal forma que sea irrelevante ejecutarlas en Linux o en el sistema operativo disponible.

En estos momentos hay un cambio tecnológico que va a obligar a rehacer los sistemas legacy, más que a mantenerlos. Esto queda claro con la aparición de la plataforma .NET de MS. Esta plataforma desecha las tecnologías impulsadas por ellos mismos (COM, COM+), de forma que las aplicaciones anteriores, desarrolladas en lenguajes como Visual Basic (que son muchas en nuestro pais), deben ser reconstruidas.

La pregunta que deben hacerse los encargados de estas aplicaciones es la siguiente: Si de todas formas es necesario que los desarrolladores aprendan un lenguaje nuevo (C#, Visual Basic.NET) por qué no aprovechar la inversión y pasar a una plataforma abierta, eliminando la dependencia de un proveedor único, el que nuevamente podría desechar unilateralmente sus tecnologías.

Capítulo 2. Por qué Java?

Existen varias formas de realizar aplicaciones multiplataformas, incluso tecnologías diseñadas originalmente para Linux se basan en técnicas multiplataformas para poder abarcar otros entornos.

Sin embargo hay otros factores que influyen en la capacidad multiplataforma de una aplicación, entre estos aspectos se pueden mencionar la conectividad a base de datos y realización de interfaces de usuario. A continuación se hará un breve analisis de las alternativas que se disponen hoy en dia para el desarrollo de aplicaciones multiplataforma en Linux.

C/GDK/GTK

El lenguaje C es bastante potente. Practicamente todo se puede hacer en C, y cuando se requiere algo demasiado especifico, siempre está la alternativa de escribir trozos en Assembly. El lenguaje de programación mayormente utilizado en Linux es C.

Existe una gran cantidad de bibliotecas para C, y a la hora de usar interfaces gráficas, GDK y GTK son bastante poderosas. El conjunto de GDK y GTK está diseñado para ser independiente de la plataforma y en cierta medida dependiente del lenguaje. Lamentablemente los ports de GDK no están tan extendidos en plataformas no Linux. Por otra parte las librerias de C específicas son generalmente ligadas a la plataforma. Por lo que habria que comenzar a combinar código generico con código especifico, lo que complica el desarrollo.

Otro aspecto a considerar es que un programador de C debe tener especial cuidado en su forma de programación. Hay hartas cosas que quedan en manos del programador, y es frecuente que surgan bugs muy dificiles de encontrar (memory leaks por ejemplo).

Por ultimo, de acuerdo al contexto de aplicaciones que se estan analizando en este trabajo, no existe una forma unificada de acceso a datos, o bien, existen intentos pero no se asegura que funcionen con todos los pares de bases de datos y arquitecturas existentes.

El lenguaje no soporta orientación a objetos en forma natural. Es posible programar orientado al objeto, pero requiere de más habilidades de parte del desarrollador.

En conclusión, es recomendable el uso de C/GDK/GTK para aplicaciones específicas en Linux, pero para desarrollo multiplataforma no es inmediatamente aplicable.

C++/QT

Esta combinación añade la ventaja de poder aplicar orientación a objetos gracias a facilidades que provee el propio lenguaje. Pero el resto de los problemas mencionados para C, se mantiene.

C#/GTK#

Como parte del proyecto .NET aparece este C remozado. El lengaje promete bastante, y es una directa competencia a Java.

C# intenta aplacar los problemas de C/C++, y define su propia libreria de clases. C# fue creado por Microsoft, pero se pueden crear implementaciones alternativas. El proyecto Mono corresponde a la implementación libre de C# y está en pleno desarrollo.

C# es una alternativa a considerar seriamente si se desean desarrollar aplicaciones multiplataforma. El unico “detalle” es que la implementación en entornos distintos a windows aun no se realiza completamente, y va a pasar un tiempo hasta que esto se haya superado.

Java

Java es un lenguaje semi orientado a objetos similar a C++. Se podría decir que es un lengueje diseñado a partir de C++ pero eliminando características que pueden complicar bastante a los programadores no expertos. Por ejemplo, la asignacion de memoria es monitoreada por el sistema, y existe un garbage collector que se encarga de reasignar la memoria de objetos que ya no estén siendo referenciados.

Otra característica de Java es que está pensado para producir código multiplataforma. Cuando se compila una fuente de Java se genera un archivo binario (bytecode) que corre en una Maquina Virtual de Java (JVM de aqui en adelante). La JVM es similar a un PC a los ojos del codigo código binario java, y le provee un entorno de ejecución completo. Esta JVM permite que las aplicaciones puedan correr en forma independiente al sistema operativo siempre que haya una JVM especificamente diseñada para el.

El mito de los applets

Sun comenzó promocionando Java como una herramienta ideal para la web. Gracias a su característica multiplataforma a nivel de codigo binario, pretendia ser una buena extension cuando el HTML no era capaz de ofrecer más.

La idea de Java en la web consistía en que los usuarios podian descargar aplicaciones ya compiladas y estás se ejecutarian en cualquier sistema operativo gracias a la Java Virtual Machine. En este contexto aparecen los applets, pequeñas aplicaciones que se incrustaban en las paginas web para añadirle funcionalidad.

Todo esto que en papel suena bonito, en la realidad no funcionaba muy bien. Los applets fueron utilizados solo para agregar funcionalidad irrelevante en la web (principalmente animaciones), lo que hacia que las paginas fueran mas lentas en descargar completamente sin que ello implicara un beneficio a nivel de funcionalidad. Por otra parte, las aplicaciones no funcionaban tan rápido y la interfaz de usuario (cuando existia) no se integraba con el sistema operativo anfitrión.

Hoy en dia es muy extraño encontrar paginas web que usen applets.

Servlets y Java en el lado del servidor

Java no es solo applets. Hoy en dia existen poderosos servidores de aplicaciones que usan Java como plataforma de ejecución. Sin entrar en demasiados detalles, Java permite tener aplicaciones distribuidas funcionando en un administrador de transacciones que se encarga de activar componentes, comunicarlos y asegurarse de que actuen en forma consistente.

Una característica muy importante de los sevidores de aplicaciones J2EE es su escalabilidad. A nivel de aplicación es transparente la forma en que se distribuyen los componentes. Estos pueden distribuirse entre distintos servidores, incluso con distinta arquitectura.

Java Hoy

Afortunadamente Java ha progresado bastante desde sus primeras incursiones en forma de applets. A la libreria de clases fundamental se han agregado un conjunto de APIs promocionadas por grupos tan importantes como el Apache group. Este soporte a nivel de API's ha logrado bajar los esfuerzos necesarios para escribir aplicaciones complejas.

En cuanto a entorno de ejecución, se han perfeccionado las JVM's y existen diversos proveedores, quienes han ido compitiendo para proveer JVM's cada vez más rápidas y eficientes.

Capítulo 3. Evolución de Java Toolkits

En el lado de los servidores las aplicaciones solo necesitan procesar datos, pero si la aplicación va a funcionar en el escritorio, debe tener una interfaz gráfica para el usuario (GUI).

En el mundo Linux se conoce bastante bien el termino de Toolkit. Un toolkit es un conjunto de controles de interfaz (widgets) que permiten simplificar el desarrollo de aplicaciones. Por ejemplo existen widgets de botones, entradas de texto, listas de selección, menués, etc. El desarrollador de aplicaciones sólo se preocupa de armar su aplicacion con un conjunto de widgets que posteriormente informarán al sistema cómo el usuario está interactuando con él, y a su vez podrá deplegar información como resultado de estas acciones.

Ejemplos de toolkits conocidos en Linux son GTK+, QT y Motif. En el caso Windows existe un toolkit nativo, y uno de un poco mas alto nivel de abstracción llamado MFC.

El uso de un toolkit standard ayuda a que varias aplicaciones creadas por distintos desarrolladores se comporte en una forma similar, de tal forma que su uso sea fácil de aprender. Ademas la adopción de un toolkit hace que una parte importante de la aplicación se base en componentes probados y estables.

AWT : Abstract Window Toolkit

Java es un entorno multiplataforma, por lo tanto en un principio se descartó el uso de toolkits nativos o específicos de una plataforma. Entonces se creó un toolkit que fuera propio de la JVM y que formara parte de la libreria de clases. A este tookit se le llamó Abstract Window Toolkit.

AWT fue el primer intento de proveer un toolkit para Java, y no se han hecho mejoras desde 1997. AWT es bastante rudimentario practicamente nadie lo consideraria en forma seria hoy en dia. Lo unico que aun es usable de AWT son sus clases más basicas que contienen elementos que son comunes a cualquier toolkit (Point, Rect, etc).

SWING/JFC

SWING es la evolución de AWT. Se podria decir que la versión actual de AWT se llama SWING.

Uno de los objetivos de crear SWING era poder enchufar un look&feel a la GUI en una forma independiente a los datos que esta contiene y en lo posible independiente a la plataforma de ejecución. Es asi como SWING tiene varios look&feel como son : Motif (Unix), Windows (Win32), y Metal (todas las plataformas).

Este toolkit utiliza a full una arquitectura del tipo Model-View-Controller. En donde se distinguen 3 componentes: uno que mantiene la estructura de datos interna (Model), otro que indica cómo esta estructura se muestra al mundo exterior (View), y finalmente un componente que permite interactuar con los anteriores mediante eventos y escuchadores de eventos.

Esta característica de SWING permite tener interfaces altamente abstractas, en donde comunmente el desarrollador solo se encarga de proveer de un modelo y SWING se encarga del resto. Tambien es posible tener varias vistas distintas de un mismo modelo.

En terminos de funcionalidad SWING es un toolkit bastante completo y a la vez complejo. La curva de aprendizaje es alta si no se tienen bien asimilados los conceptos de M-V-C, pero una vez pasada esta etapa el desarrollo de aplicaciones con GUI con Java se convierte en una opcion bastante potente.

Similar a lo que sucedió con los applets, todo esto que parece muy bueno tiene sus inconvenientes en la vida real:

- Las aplicaciones que usan SWING necesitan una buena cantidad de RAM para funcionar

- Los tiempos de respuesta o feedback para el usuario son notablemente lentos, en comparación a los toolkits nativos
- Las aplicaciones SWING, sobre todo con el L&F Metal, son distintas a sus aplicaciones nativas, esto produce confusión a los usuarios

SWT : Standard Window Toolkit

Este Toolkit nace en el contexto del proyecto Eclipse[1]. Este projecto pretende ser un framework para construir IDE's (Integrated Development Environment) para distintos tipos de herramientas, basado en una arquitectura de plugins. Eclipse fue desarrollado inicialmente por IBM y se liberó posteriormente bajo licencia CPL.

El toolkit SWT surge como necesidad de cubrir los problemas de SWING, a nivel de dieño fundamental. A pesar de que se pueden usar look&feel's similares a un entorno nativo, estos siguen siendo simulados por SWING, consumiendo recuros innecesarios, ralentizando la interfaz y provocando "ruido" en cuanto a consistencia de interfaz se refiere.

El objetivo de SWT es crear un toolkit que sea nativo y a la vez portable. Estas dos características que parecen ser imposibles de cumplir en forma conjunta, son posibles gracias a la API JNI : Java Native Interface. JNI permite que una aplicación Java interactue con código nativo de la plataforma, por ejemplo bibliotecs .so en Linux.

A ojos del desarrollador, SWT es una API que representa a un toolkit multiplataforma, en donde encuentra toda la funcionalidad típica de los toolkits. Desde el punto de vista de la implementación de SWT, éste es una capa muy delgada entre Java y las librerias nativas del toolkit de la plataforma.

SWT siempre trata de hacer un mapeo uno a uno entre el toolkit nativo y el abstracto. Cuando una característica no está disponible en el toolkit nativo, SWT lo emula.

Existen varios ports de SWT para las plataformas más populares como:

- Linux/GTK+ 2.x
- Linux/Motif
- Windows/MFC
- Solaris, AIX, HP-UX / Motif
- QNX/Photon

A diferencia de SWING, el toolkit SWT ocupa muy pocos recursos, es notablemente más veloz y se integra perfectamente con el toolkit nativo. Podemos tener, por ejemplo, una aplicacion Java corriendo en Linux con una interfaz gráfica GTK+. Esta misma aplicacion, sin cambios puede correr en un entorno Windows pero usará la interfaz nativa Win32.

Notas

1. http://www.eclipse.org

Capítulo 4. Desarrollo de aplicaciones con SWT

Para desarrollar con SWT se debe bajar un grupo de bibliotecas del proyecto Eclipse. Hay archivos distintos para cada plataforma, y se debe bajar una versión especifica. Aunque el archivo es bastante grande, la parte que nos interesa es pequeña.

Si vemos el directorio de Eclipse, hay un subdirectorio `plugins` en donde se encuentran todos los plugins que conforman Eclipse. El que nos interesa corresponde a `org.eclipse.swt`. En ese directorio encontraremos a su vez dos subdirectorios:

- os : Aqui se encuentran las bibliotecas específicas del sistema operativo. Basicamente son bibliotecas nativas que conectan a Java con las bibliotecas del toolkit nativo a traves de JNI

  En el caso de Linux encontraremos los subdirectorios linux/x86 y un conjunto de bibliotecas `libswt*.so`

- ws : En este directorio encontraremos la implementacion en Java de SWT asociado a las bibliotecas del directorio `os`. En la version Linux encontraremos un directorio `gtk1x` que contiene bibliotecas de clases java `swt*.jar` y los fuentes Java de estas clases

  Nota: El proyecto eclipse soporta actualmente solo GTK2, los ejemplos indicados corresponden a la version GTK1

Aqui hay una muestra del contenido completo del directorio `org.eclipse.swt`

```
[fcatrin@shaman org.eclipse.swt]$tree
.
|-- about.html
|-- os
|   `-- linux
|       `-- x86
|           |-- about.html
|           |-- cpl-v05.html
|           |-- lgpl-v21.txt
|           |-- libswt-gtk-2034.so
|           |-- libswt-pi-1x-gtk-2034.so
|           `-- libswt-pixbuf-1x-gtk-2034.so
|-- plugin.properties
|-- plugin.xml
`-- ws
    `-- gtk1x
        |-- swt-pi.jar
        |-- swt-pi.jar.bin.log
        |-- swt-pisrc.zip
        |-- swt.jar
        |-- swt.jar.bin.log
        `-- swtsrc.zip

5 directories, 15 files
```

El clasico Hola Mundo

Hacer aplicaciones con SWT es bastante directo y sencillo. Si ya se tiene experiencia con SWING e incluso con GTK, la forma de trabajo será muy familiar.

A continuación, el clasico "Hello Word" en SWT

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
```

```
public class HelloWorld {

 public static void main(String[] args) {
  Display display = new Display();

  Shell shell = new Shell(display);
  shell.setText("HelloWorld");

  FillLayout layout = new FillLayout();
  layout.type = SWT.VERTICAL;
  shell.setLayout(layout);

  Label label = new Label(shell, SWT.CENTER);
  label.setText("Hello World");

  shell.open();
  while (!shell.isDisposed())
   if (!display.readAndDispatch()) display.sleep();
 }
}
```

Este código crea una ventana con el caption "HelloWorld" e inserta una etiqueta al interior de la aquella. Luego, como toda aplicacion basada en eventos, se queda en un ciclo despachando eventos hasta que se cierra la ventana.

Y ya tenemos una aplicacion que ocupa el toolkit nativo, pero multiplataforma.

Widgets y Layouts

En SWT se pueden encontrar varios widgets comunes : botones, entradas de texto, listas, etc. Se puede encontrar la referencia completa en la Documentación de Eclipse[1]

Otro aspecto a destacar es que la disposición de los componentes se realiza en base a layouts. El desarrollador no se tiene que preocupar de dimensiones y ajustes a menos que sea realmente necesario. Los widgets se van empaquetando en distintos tipos de layouts, al igual que GTK se disponen layouts verticales, horizontales y tablas, pero en SWT se agrega la posibilidad de crear layouts personalizados extendiendo las clases provistas.

Una aplicacion simple de ejemplo

Para aprender SWT aun no existen tutoriales completos que uno podria recomendar. En caso de dudas con la referencia de SWT, se puede ir a mirar el código de Eclipse.

Si el código de Eclipse es muy grande, tambien se puede ver una aplicacion bastante simple que se encuentra en desarrollo. Se trata de SQLAdmin, aplicación GPL que puede ser descargada desde http://sqladmin.sourceforge.net.

SQLAdmin es un cliente SQL standard, que permite conectarse a cualquier Base de Datos que tenga driver JDBC (practicamente todas).

Notas

1. http://www.eclipse.org/documentation/html/plugins/org.eclipse.platform.doc.isv
2. http://sqladmin.sourceforge.net

Capítulo 5. GCJ : GNU Compiler for Java

Una aplicacion Java con SWT funciona bastante mejor que cualquier aplicacion para escritorio que hayamos visto con SWING o AWT, pero aun podemos llegar más allá.

Como ya se habia mencionado, una aplicación Java corre en una JVM, que es una implementación de una maquina por software. Esta JVM se encarga de convertir los bytecodes a algo que se pueda ejecutar en la maquina real. Esto que es una gran ventaja desde el punto de vista de la programación, es una desventaja a la hora de medir la eficiencia y el rendimiento de la aplicación.

Una forma de mejorar esto es realizar Just In Time Compiling o JITC. Con esta técnica el codigo primero Java es pasado a código nativo y luego ejecutado como tal. En vez de ir instrucción por instrucción, se junta un grupo funcional y se ejecuta.

Las JVM con JITC han significado un avance, pero aun hay una desventaja frente a usar código puro. Y es en este contexto en donde aparece GCJ : GNU Compiler for Java.[1]

Java en Código Nativo

Desde el punto de vista del uso de memoria, en el esquema tradicional se desperdicia memoria, por la carga de la JVM y porque ésta necesita mantener su propio estado de operación al ejecutar Java bytecodes. Por ejemplo el uso JITC requiere que los resultados de la compilacion se vayan almacenando para su uso posterior. Se podria argumentar que la memoria es barata en estos dias, pero cuando se tienen varias aplicaciones corriendo en la misma máquina, hasta una máquina grande puede verse en problemas.

GCJ es un compilador que permite convertir código fuente Java en código nativo. Incluso es capaz de convertir codigo binario java (.class) en código nativo. El código nativo de varias clases se linkea para formar una unica aplicación nativa.

En cuanto a rendimiento, obviamente hay un cambio mas crítico, aunque no hay mediciones suficientes, se puede esperar un aumento de rendimiento entre un 10% a 15% respecto a JITC.

Java Class Libraries

Una aplicación Java no puede funcionar por si sola. Hasta en las cosas mas simples se apoya en bibliotecas de clases que son provistas por la JVM. Estas bibliotecas son simplemente más clases Java que pueden ser puras o interactuan con bibliotecas de mas bajo nivel (.so por ejemplo).

Una de las dificultades que ha tenido que enfrentar el proyecto GCJ es convertir estas bibliotecas de clases en código nativo, para que las aplicaciones nativas puedan ser linkeadas con ellas.

La limitación que existe es debido a que algunas clases están intimamente ligadas a la JVM, y usan funciones que no están documentadas. Debido a eso, no se dispone de la biblioteca de clases completa, pero si de gran parte de ella.

JNI y SWT

El codigo nativo generado por GCJ puede usar otro codigo nativo. A nivel de codigo fuente está realizado mediante JNI, pero esta interacción es natural.

Gracias a tecnologías como JNI y SWT podemos tener aplicaciones nativas generadas a traves de un lenguaje robusto, potente y con una gran cantidad de API's o bibliotecas disponibles. En el articulo "Create native, cross-platform GUI applications"[2] se puede ver un ejemplo de SWT con GCJ

Ventaja para Linux

El desarrollo de GCJ se ha orientado principalmente a Linux, y esto lo pone en ventaja respecto a otros sistemas operativos. Cuando GCJ es aplicable podemos tener aplicaciones mas pequeñas y rapias a partir del mismo código fuente.

Desde el punto de vista de SWT, el aporte de IBM ha sido indiscutido. En un principio habia SWT solo para Motif en Linux, pero gracias a tratarse de un proyecto abierto, se agregó posteriormente el soporte para GTK debido a la solicitud de los mismos usuarios. Es mas, desde mucho antes que GTK2 fuera liberado oficialmente con GNOME2, el equipo de Eclipse ya lo habia adoptado como toolkit oficial de la serie GTK.

Perspectivas

El futuro de las herramientas de desarrollo para Linux se ve bastante promisorio. No solo en el ambito de Java, sino tambien con la aparición de alternativas similares (C#).

Un desarrollador que venga desde un entorno no Linux ahora se va a encontrar con herramientas que le son conocidas, y debido a la potencia de éstas, las excusas de no tener aplicaciones de escritorio para Linux se van agotando. Perfectamente se puede desarrollar con esta tecnología y ejecutar en donde se desee.

El proceso ha tomado varios años, y han habido falsas alarmas (applets, awt, swing), pero se podria decir que al fin es posible la independencia de la plataforma, sin sacrificar otros aspectos importantes como el uso de recursos y el rendimiento de las aplicaciones.

Notas

1. http://gcc.gnu.org/java/
2. http://www-106.ibm.com/developerworks/linux/library/j-nativegui/index.html?dwzone=linux