O - Unidad didáctica O. Unidad introductoria

O.I - Presentación y definición del software libre

O.I.I - Historia y definición del software libre.

Un poco de historia...

Entre los años 60 y 70 del siglo XX, el software no era considerado un producto sino un añadido que los vendedores de los grandes computadores de la época (los llamados **mainframes**) aportaban a sus clientes para que éstos pudieran usarlos.

Figura 1: Mainframe IBM S/360 Modelo 91, desarrollado en Centro de Desarrollo de IBM (Böblingen)

En dicha cultura, era común que los programadores y desarrolladores de software compartieran libremente sus programas unos con otros. Este comportamiento era particularmente habitual en algunos de los mayores grupos de usuarios de la época, como **DECUS** (grupo de usuarios de computadoras DEC). A finales de los 70, las compañías iniciaron el hábito de imponer restricciones a los usuarios, con el uso de acuerdos de licencia.

Definición de software libre (*free software*)

El termino **free**, traducido al castellano, significa tanto "libre" como "gratis", por eso muchas veces suelen confundirse el freeware con el software libre aunque entre ambos existen notables diferencias, que detallamos a continuación.

Software libre (en inglés ***free software***) es el software que, una vez obtenido, puede ser usado, copiado, estudiado, modificado y redistribuido libremente. El software libre suele estar disponible de forma gratuita en la red Internet o a precio del coste de la distribución a través de otros medios.

Entonces, ¿el software libre siempre es software gratuito? No es obligatorio que sea así y, aunque conserve su carácter de libre, puede ser vendido comercialmente. Análogamente, el software gratuito (denominado usualmente ***freeware***) incluye en algunas ocasiones el código fuente. Pero a diferencia del software libre, el freeware **no es libre** en el mismo sentido que el software libre, al menos que se garanticen los derechos de modificación y redistribución de dichas versiones modificadas del programa.

Por otro lado, no debe confundirse software libre con software de **dominio público**. Éste último es aquél por el que no es necesario solicitar ninguna licencia y cuyos derechos de explotación son para toda la humanidad, porque pertenece a todos por igual. Cualquiera puede hacer uso de él, siempre con fines legales y consignando su autoría original. Este software sería aquél cuyo autor lo dona a la humanidad o cuyos derechos de autor han expirado. Si un autor condiciona su uso bajo una licencia, por muy débil que sea, ya no es dominio público.

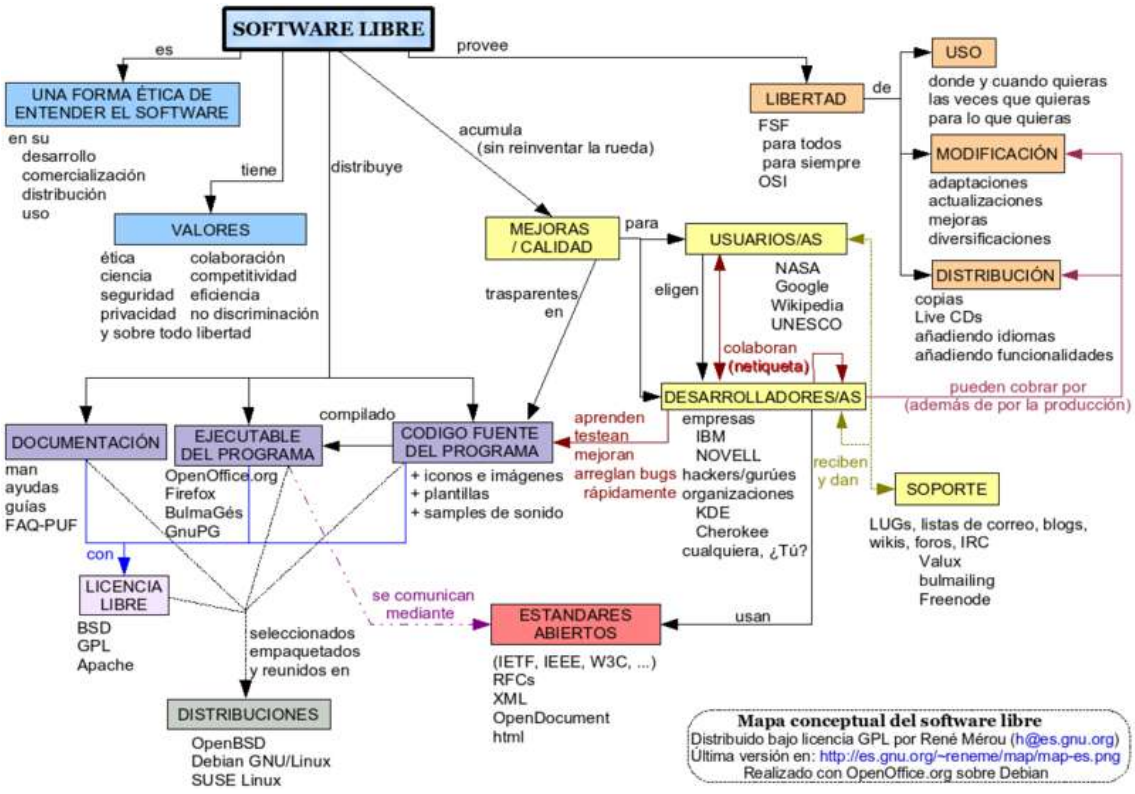


Figura 2: Mapa conceptual del software libre

Libertades del software libre

De acuerdo con la definición antes expuesta, el software es "libre" si garantiza las siguientes libertades:

- **Libertad 0:** ejecutar el programa con cualquier propósito (privado, educativo, público, comercial, etc.)
- **Libertad 1:** estudiar y modificar el programa (para lo cual es necesario poder acceder al código fuente)
- **Libertad 2:** copiar el programa de manera que se pueda ayudar al vecino o a cualquiera
- **Libertad 3:** mejorar el programa, y hacer públicas las mejoras, de forma que se beneficie toda la comunidad

Es importante señalar que las libertades 1 y 3 obligan a que se tenga acceso al código fuente. La **"Libertad 2"** hace referencia a la libertad de modificar y redistribuir el software libremente licenciado bajo algún tipo de licencia de software libre que beneficie a la comunidad.

Tipos de licencias

Una **licencia** es aquella autorización formal con carácter contractual que un autor de un software da a un interesado para ejercer "actos de explotación legales". Pueden existir tantas licencias como acuerdos concretos se den entre el autor y el licenciatario. Desde el punto de vista del software libre, existen distintas variantes del concepto o grupos de licencias:

- Las libertades definidas anteriormente están protegidas por licencias de software libre, de las cuales una de las más utilizadas es la **Licencia Pública General GNU** (**GPL**). El autor conserva los derechos de autor (copyright), y permite la redistribución y modificación bajo términos diseñados para asegurarse de que todas las versiones modificadas del software permanecen bajo los términos más restrictivos de la propia GNU GPL. Esto hace que no sea imposible crear un producto con partes no licenciadas GPL: el conjunto tiene que ser GPL.

- **Licencias BSD**, llamadas así porque se utilizan en gran cantidad de software distribuido junto a los sistemas operativos BSD. El autor, bajo tales licencias, mantiene la protección de copyright únicamente para la renuncia de garantía y para requerir la adecuada atribución de la autoría en trabajos derivados, pero permite la libre redistribución y modificación, incluso si dichos trabajos tienen propietario. Son muy permisivas, tanto que son fácilmente absorbidas al ser mezcladas con la licencia GNU GPL con quienes son compatibles.

Puede argumentarse que esta licencia asegura "verdadero" software libre, en el sentido que el usuario tiene libertad ilimitada con respecto al software, y que puede decidir incluso redistribuirlo como no libre. Otras opiniones están orientadas a destacar que este tipo de licencia no contribuye al desarrollo de más software libre.

- **Licencias estilo MPL y derivadas**: esta licencia es de software libre y tiene un gran valor porque fue el instrumento que empleó Netscape Communications Corp. para liberar su Netscape Communicator 4.0 y empezar ese proyecto tan importante para el mundo del software libre: el navegador web Mozilla.

  Se utilizan en gran cantidad de productos de software libre de uso cotidiano en todo tipo de sistemas operativos. La MPL es software libre y promueve eficazmente la colaboración evitando el efecto "viral" de la GPL (si usas código licenciado GPL, tu desarrollo final tiene que estar licenciado GPL). Desde un punto de vista del desarrollador la GPL presenta un inconveniente en este punto, y lamentablemente mucha gente se cierra en banda ante el uso de dicho código. No obstante la MPL no es tan excesivamente permisiva como las licencias tipo BSD.

  Estas licencias son denominadas ***de copyleft débil***. La **NPL** (luego la MPL) fue la primera licencia nueva después de muchos años, que se encargaba de algunos puntos que no fueron tenidos en cuenta por las licencias BSD y GNU. En el espectro de las licencias de software libre se le puede considerar adyacente a la licencia estilo BSD, pero perfeccionada.

Hay que hacer constar que el titular de los derechos de autor (copyright) de un software bajo licencia copyleft puede también realizar una versión modificada bajo su copyright original, y venderla bajo cualquier licencia que desee, además de distribuir la versión original como software libre. Esta técnica ha sido usada como un modelo de negocio por una serie de empresas que realizan software libre, pues esta práctica no restringe ninguno de los derechos otorgados a los usuarios de la versión copyleft.

También podría retirar todas las licencias de software libre anteriormente otorgadas, pero esto obligaría a una indemnización a los titulares de las licencias en uso. En España, toda obra derivada está tan protegida como una original, siempre que la obra derivada parta de una autorización contractual con el autor. En el caso genérico de que el autor retire las licencias "copyleft", no afectaría de ningún modo a los productos derivados anteriores a esa retirada, ya que no tiene efecto retroactivo. En términos legales, el autor no ha derecho a retirar el permiso de una licencia en vigencia. Si así sucediera, el conflicto entre las partes se resolvería en un pleito convencional

0.1.2 - Motivaciones para su creación y sostenibilidad.

Una de las características del software libre es no solamente que el usuario tiene libertad para modificar el código -- adaptarlo a sus necesidades específicas --, sino también haber difundido masivamente un modelo de desarrollo cooperativo y comunitario del software, que se revela hoy mucho más eficiente que la lógica propietaria de las grandes empresas del sector.

El software libre es ante todo una cuestión de **libertad** y de **comunidad**. El software libre incentiva la cooperación entre usuarios y desarrolladores libremente. Esta libertad tiene su consecuencia en una mejora constante de la calidad de las distribuciones y aplicaciones desarrolladas, de los cual se benefician directamente los usuarios finales.

Además el uso extendido de aplicaciones de software libre supone una reducción de costes para las empresas, que se evitan unas fuertes inversiones en tecnología.

0.1.3 – Comparación con el software propietario.

Una vez analizadas las características principales del software libre, vamos a ver los aspectos que determina el software no libre o propietario.

El **software no libre** (también llamado software propietario, software privativo, software privado, software con propietario o software de propiedad) se refiere a cualquier programa informático en el que los usuarios tienen limitadas las posibilidades de usarlo, modificarlo o redistribuirlo (con o sin modificaciones), o cuyo código fuente no está disponible o el acceso a éste se encuentra restringido.

Para la Fundación de Software Libre (**FSF**) este concepto se aplica a cualquier software que no es libre o que sólo lo es parcialmente (semilibre), bien porque su uso, redistribución o modificación está prohibida, o bien porque requiere permiso expreso del titular del software.

En el software no libre una persona física o jurídica (compañía, corporación, fundación, etc.) posee los derechos de autor sobre un software negando o no otorgando, al mismo tiempo, los derechos de usar el programa con cualquier propósito, de estudiar cómo funciona el programa y adaptarlo a las propias necesidades (donde el acceso al código fuente es una condición previa), de distribuir copias o de mejorar el programa y hacer públicas las mejoras (para esto el acceso al código fuente es un requisito previo).

De esta manera, un programa sigue siendo no libre aún si el código fuente se hecho público, cuando se mantiene la reserva de derechos sobre el uso, modificación o distribución (por ejemplo, el programa de licencias ***shared source de Microsoft***).

0.1.4 – Beneficios del código abierto

El código abierto (en inglés, ***open source)*** posee una serie de beneficios, entre los cuales destacamos:

- Ahorro considerable en inversión necesaria en aplicaciones software
- Las herramientas open source son capaces de proporcionar la misma funcionalidad que las versiones alternativas de pago.
- Siempre está sujeto a contínuas revisiones y mejoras, como consecuencia del creciente nº de usuarios que desarrollan y depuran estas aplicaciones.
- Proporciona libertad e independencia tecnológica a nivel personal e institucional.

Figura 3: Logotipo representativo de código abierto (open source)

0.2 El sistema operativo GNU Linux

0.2.1 - Historia de Linux

El proyecto GNU

El **proyecto GNU** fue iniciado por Richard Stallman con el objetivo de crear un sistema operativo completo libre: el **sistema GNU**. El 27 de Septiembre de 1983 se anunció públicamente el proyecto por primera vez en el grupo de noticias net.unix-wizards. Al anuncio original, siguieron otros ensayos escritos por Richard Stallman como el "Manifesto GNU", que establecieron sus motivaciones para realizar el proyecto GNU, entre las que destaca "volver al espíritu de cooperación que prevaleció en los tiempos iniciales de la comunidad de usuarios de computadoras". GNU es un acrónimo recursivo que significa "**GNU no es Unix**". Stallman sugiere que se pronuncie, en inglés como "guh-noo" (se puede observar que el logo es un ñú) para evitar confusión con "new" (nuevo). En español, GNU se pronuncia fonéticamente.

Figura 4: Logotipo representativo del proyecto GNU

Acerca de Unix...

UNIX es un sistema operativo no libre muy popular, porque está basado en una arquitectura que ha demostrado ser técnicamente estable. El sistema GNU fue diseñado para ser totalmente compatible con UNIX. El hecho de ser compatible con la arquitectura de UNIX implica que GNU esté compuesto de pequeñas piezas individuales de software, muchas de las cuales ya estaban disponibles, como el sistema de edición de textos TeX y el sistema gráfico X-Window, que pudieron ser adaptados y reutilizados.

Para asegurar que el software GNU permaneciera libre para que todos los usuarios pudieran "ejecutarlo, copiarlo, modificarlo y distribuirlo", el proyecto debía ser liberado bajo una licencia diseñada para garantizar esos derechos al tiempo que evitase restricciones posteriores de los mismos. La idea se conoce en inglés como **copyleft** – "izquierdo de copia" - (en clara oposición a **copyright** – "derecho de copia"), y está contenida en la Licencia General Pública de GNU (GPL).

El sistema operativo GNU/Linux

GNU/Linux (GNU con Linux) es la denominación defendida por Richard Stallman y otros para el sistema operativo que utiliza el núcleo (kernel) Linux en conjunto con las aplicaciones de sistema creadas por el proyecto GNU. Comúnmente este sistema operativo es denominado como Linux, aunque según Stallman esta denominación no es correcta.

Figura 5: Richard Matthew Stallman. Programador y creador del proyecto GNU.

Desde 1984, Richard Stallman y voluntarios están intentando crear un sistema operativo libre con un funcionamiento similar al Unix, recreando todos los componentes necesarios para tener un sistema operativo funcional que se convertiría en el sistema operativo GNU. En el comienzo de los años 90, después de seis años, GNU tenía muchas herramientas importantes listas, como compiladores, depuradores, intérpretes de órdenes etc. excepto por el componente central: el núcleo.

Con el surgimiento del kernel Linux, esta laguna fue llenada y surgió el sistema operativo con el kernel Linux en conjunto con las herramientas GNU. De esta manera, Stallman juzga que este sistema operativo es una "versión modificada" del sistema GNU y por lo tanto debe tener la denominación GNU/Linux. Esta denominación resolvería la confusión entre el núcleo y el sistema operativo completo a que puede llevar, y de hecho ha llevado, la denominación Linux en solitario. Stallman también espera que con el aporte del nombre GNU, se dé al proyecto GNU que él encabeza el reconocimiento que merece por haber creado las aplicaciones de sistema imprescindibles para ser un sistema operativo compatible con UNIX.

Algunas distribuciones apoyan esta denominación, e incluyen los términos GNU/Linux en sus nombres, tal es el caso de Debian GNU/Linux o GNU/LinEx.

Algunos sectores de la comunidad de usuarios del sistema operativo han rechazado la denominación GNU/Linux por varias razones, entre ellas que ya se había empezado a denominar Linux al sistema operativo antes de que Richard Stallman promocionase esta denominación. Otras personas se oponen a la postura ideológica de Stallman radicalmente en contra del software no libre y por ello son contrarios al uso de este nombre para evitar la promoción de las ideas del fundador del proyecto GNU. Otros sectores de la comunidad han reconocido la conveniencia de este nombre.

Hay que señalar que, al igual que es una simplificación denominar al sistema que usa el usuario final Linux, obviando las aplicaciones GNU que completan el sistema operativo, el conjunto GNU + Linux representa solamente una parte (aunque importante) del software encontrado en una distribución Linux. Existe una gran cantidad de software original producido independientemente de los proyectos GNU y Linux por otras personas u organizaciones, como por ejemplo Apache, KDE, Samba u OpenOffice.org entre otros.

Linux

Linux es la denominación de un sistema operativo y el nombre de un núcleo. Es uno de los paradigmas del desarrollo de software libre (y de código abierto), donde el código fuente está disponible públicamente y cualquier persona, con los conocimientos informáticos adecuados, puede libremente estudiarlo, usarlo, modificarlo y redistribuirlo.

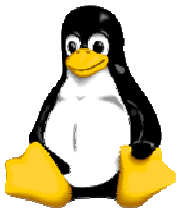


Figura 6: TuX, el logo y mascota de Linux

El término Linux estrictamente se refiere al núcleo Linux, pero es más comúnmente utilizado para describir al sistema operativo tipo Unix, que utiliza primordialmente filosofía y metodologías libres (también conocido como GNU/Linux) y que está formado mediante la combinación del núcleo Linux con las bibliotecas y herramientas del proyecto GNU y de muchos otros proyectos/grupos de software libre y no libre). El núcleo no es parte oficial del proyecto GNU (el cual posee su propio núcleo en desarrollo, llamado ***Hurd***), pero es distribuido bajo los términos de la licencia GNU GPL.

La expresión Linux también es utilizada para referirse a las distribuciones GNU/Linux, colecciones de software que suelen contener grandes cantidades de aplicaciones además del núcleo (entornos gráficos, suites ofimáticas, servidores web, etc.). Coloquialmente se aplica el término Linux a éstas, aunque en estricto rigor sea incorrecto.

La marca Linux pertenece a Linus Torvalds y se define como "*un sistema operativo para computadoras que facilita su uso y operación*". Actualmente Linus supervisa el uso (o abuso) de la marca a través de la organización sin fines de lucro Linux International.

Figura 7: Linus Torvalds, creador del núcleo de Linux.

0.2.2 – Utilidades y aplicaciones

REDES Y CONECTIVIDAD		
TIPO DE APLICACIÓN	**Windows**	**Linux**
Navegadores web	- Internet Explorer - Netscape / Mozilla - Opera - Phoenix para Windows	- Netscape, Mozilla - Galeon - Konqueros - Opera - Phoenix - Nautilus - Epiphany - Links - Dillo
Navegadores web en modo consola	Lynx para Windows	- Links - Lynx - w3m - Xemacs
Clientes de e-mail	- Outlook Express - Mozilla - Eudora - Becky	- Evolution - Netscape / Mozilla - Sylpheed, Sylpheed-claws - Kmail - Gnus - Balsa - Arrow - Gnumail - Althea
Lectores de noticias (news readers)	- Xnews - Outlook - Netscape / Mozilla	- Knode - Pan - NewsReader - Netscape / Mozilla - Pine - Mutt - tin - Xemacs
Gestor de descargas	- FlashGet - Go!zilla - Reget - Getright - Wget para Windows	- Downloader for X - Caitoo - Prozilla - Wget - Aria - Axel
Clientes FTP	- FTP in far - SmartFTP - CuteFTP	- Gftp - Kbear - IglooFTP - Nftp - Wxftp
Clientes IRC	- Mirc - Klient - VIRC - Xircon - Pirch	- Xchat - KVirc - Irssi - BitchX - Epic
Mensajería local con máquinas	WinPopup	

Windows		- LinPopup - Kpopup
Clientes para mensajería instantánea	- ICQ - MSN - AIM - Trillian ICQ	- Licq - Alicq - Gaim - Kopete - Everybuddy - aMSN
Conferencias audio / vídeo	Netmeeting	Gnomemeeting
Comunicación por voz	Speak Freely	- Speak Freely for Unix - TeamSpeak
Firewall (cortafuegos)	- BlackICE - ATGuard - ZoneAlarm - Agnitum Outpost - Winroute PRO	- Kmyfirewall - Firewall builder - Shorewall - Guarddog - Firestarter

TRABAJO CON FICHEROS

TIPO DE APLICACIÓN	Windows	Linux
Administrador de Archivos al estilo FAR y NC	- FAR - Norton Commander	- Midnight Commander - X Northern Captain - Deco - Portos Commander - Konqueror
Administrador de archivos al estilo Windows	Windows Explorer	- Konqueror - Gnome-Commander - Nautilus - XWC
Inspección rápida de ficheros HTML locales	Internet Explorer	- Dillo - Konqueror - Nautilus - Lynx / Links

SOFTWARE DE ESCRITORIO

TIPO DE APLICACIÓN	Windows	Linux
Editor de textos	Ntepad, Wordpad, Textpad	- KEdit - GEdit - Gnotepad - Kate - Kwrite - Vim - Xemacs
Editor de texto modo consola	FAR Editor	- Vim - Emacs - pco - je - Jed
Compresores	Winzip, Winrar	- FileRoller - Gnozip - Linzip - RAR for Linux
Antivirus	AVG, Dr. Web, Trendmicro	- Dr. Web for Linux - RAV Antivirus - OpenAntivirus - VirusHammer
Configuración del sistema	Msconfig	- Linuxconf - Webmin - Yast, Yast2 - RAR for Linux
Software para backup	- ntbackup - Legato Networker	- Lonetar - Disk archive - Bacula Taper
Administrador de tareas	Taskman, Taskinfo	- Top - Gtop, Ktop - kSysGuard
Reconocimiento de voz	ViaVoice	Sphinx
Recuperación de datos	R-Studio	- e2undel - myrescue - TestDisk

MULTIMEDIA

TIPO DE APLICACIÓN	Windows	Linux
Reproductores de música (mp3, ogg)	Winamp	- XMMS - Noatun - Zinf - SnackAmp
Grabación de CDs, DVDs	Nero, Roxio, Easy CD Creator	- K3b - XCDRoast - KOnCd - GCombust - WebCDWriter
Reproductores de CD	CD Player	- ksCD - Oprheus - Sadp - Workman
Decodificadores MP3	Lame	- Lame - Bladeenc - NotLame - gogo
Editores de audio	Soundforge, Cooledit	- Sweep - Waveforge - Sox - Audacity
Secuenciador MIDI	CakeWalk	- RoseGarden - Brahms - Anthem - Melys

DISEÑO GRÁFICO – RETOQUE FOTOGRÁFICO

TIPO DE APLICACIÓN	Windows	Linux
Visualizador de archivos gráficos	ACDSee, Irfanwiew	- Xnview - GQView - Qiv - Compupic
Editores simples	Paint	- Kpaint - TuxPaint
Editores complejos	Adobe Photoshop	The Gimp
Secuenciador MIDI	CakeWalk	- RoseGarden - Brahms - Anthem - Melys

0.2.3 – Versiones existentes

Linux es un sistema de libre distribución, por lo que podemos encontrar todos los ficheros y programas necesarios para su funcionamiento en multitud de servidores conectados a Internet.

La tarea de reunir todos los ficheros y programas necesarios, así como instalarlos en nuestro sistema y configurarlos puede resultar una tarea bastante complicada. Debido a esto nacieron las llamadas **distribuciones de Linux** (**Linux distros**), es decir, empresas y organizaciones que se dedica a hacer el trabajo "sucio" para nuestro beneficio y comodidad.

Una distribución no es otra cosa, que una recopilación de programas y ficheros, organizados y preparados para su instalación. Estas distribuciones se pueden obtener a través de Internet, o comprando los CDs de las mismas, los cuales contendrán todo lo necesario para instalar un sistema Linux bastante completo y en la mayoría de los casos un programa de instalación que nos ayudara en la tarea de una primera instalación. Casi todos los principales distribuidores de Linux, ofrecen la posibilidad de bajarse sus distribuciones, via FTP (sin cargo alguno).

Existen muchas y variadas distribuciones creadas por diferentes empresas y organizaciones a unos precios bastante asequibles.

A continuación se muestra información relativa a las distribuciones más importantes de Linux (aunque no las únicas):

DISTRIBUCIONES LINUX		
	Debian	Distribución con muy buena calidad. El proceso de instalación es quizás un poco mas complicado, pero sin mayores problemas. Gran estabilidad antes que últimos avances.
	RedHat	Esta es una distribución que tiene muy buena calidad, contenidos y soporte a los usuarios por parte de la empresa que la distribuye. Es necesario el pago de una licencia de soporte. Enfocada a empresas.
	Fedora	Esta es una distribución patrocinada por RedHat y soportada por la comunidad. Fácil de instalar.
	Gentoo	Esta distribución es una de las únicas que últimamente han incorporado un concepto totalmente nuevo en Linux. Es un sistema inspirado en BSD-ports. Se puede compilar/optimizar el sistema completamente desde cero. No es recomendable adentrarse en esta distribución sin una buena conexión a Internet, un ordenador medianamente potente y cierta experiencia en sistemas Unix.
	SuSe	Otra de las grandes. Calidad germana. Fácil de instalar.

Existen además unas versiones de las distintas distribuciones de Linux, llamadas **LiveCD** que sirven para probar como funciona un sistema Linux, sin necesidad de instalaciones ni espacio libre en el disco duro.

Un LiveCD no es otra cosa que una distribución de Linux que funciona al 100%, sin necesidad de instalarla en el ordenador donde la probamos. Utiliza la memoria RAM del ordenador para "instalar" y arrancar la distribución en cuestión. En la memoria también se instala un "disco virtual" que emula al disco dure de un ordenador.

De esta forma sólo hace falta introducir el CD o DVD en el ordenador en cuestión y arrancarlo, al cabo de unos minutos tendremos un sistema Linux funcionando en el mismo. Este tipo de distribuciones solamente sirve para demostraciones y pruebas, ya que una vez que apagamos el ordenador, todo lo que hemos hecho desaparece.

Algunas distribuciones tipo "LiveCD" vienen también con la opción de instalación una vez que la hemos probado.

Por último se ofrece una gráfica con las distribuciones más comunes, ordenadas cronológicamente por orden de aparición en el mercado, desde el año 1991 hasta hoy.

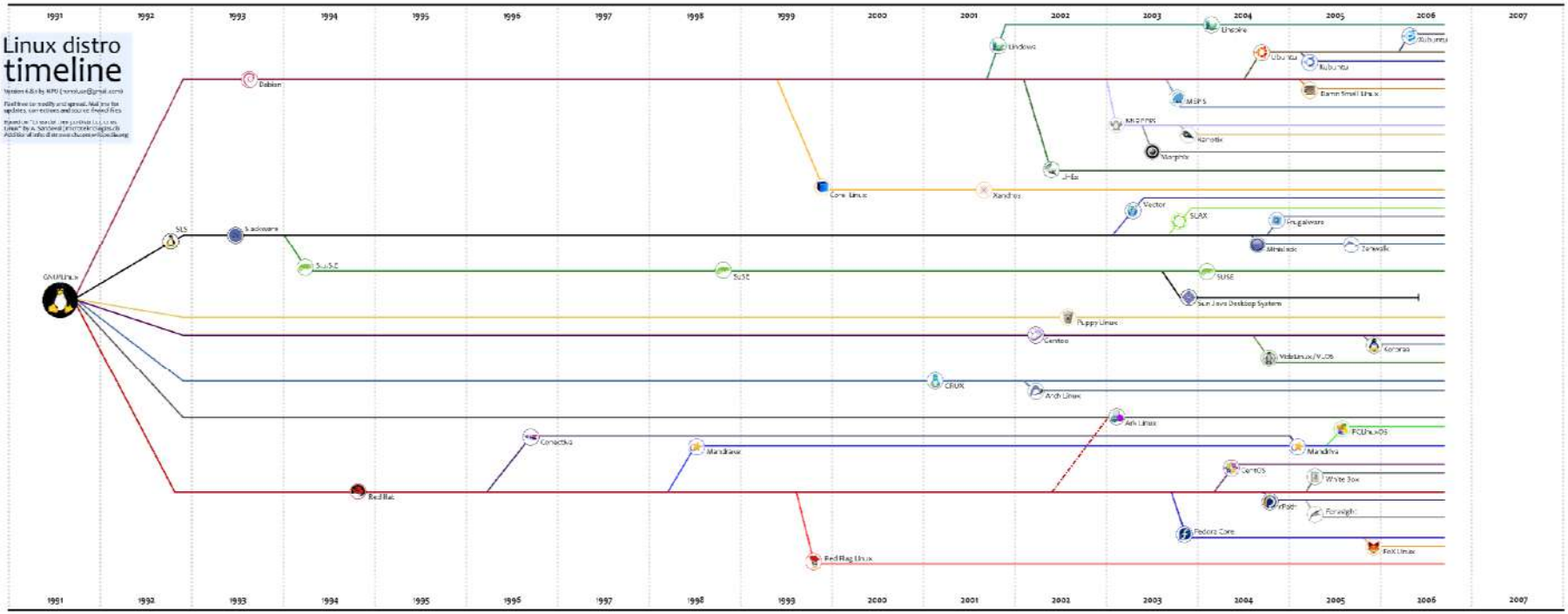


Figura 8: Cronología de distribuciones Linux (1991 – 2007)

0.3 – Guadalinex, la distribución andaluza

0.3.1 – Características de esta distribución.

Guadalinex es una distribución Linux promovida por la Junta de Andalucía para fomentar el uso del software libre dentro de Andalucía. Está inspirada en un proyecto similar de la Junta de Extremadura: GnuLinEx. Estuvo basada en la distribución Debian GNU/Linux gracias al acuerdo inicial con la Junta de Extremadura, si bien desde la versión actual (**v3**) se basa en Ubuntu.

Entre las características más relevantes de esta distribución destacamos:

- Incluye todos los programas habituales para el uso habitual del ordenador como: aplicaciones de oficina, Internet, diseño gráfico, multimedia, juegos...
- Se puede probar sin necesidad de instalación, como LiveCD.
- También es posible instalarla en el disco duro, como sistema operativo único o compartiendo disco con otros sistemas cualesquiera.
- Cuenta con el respaldo de la Junta de Andalucía y con una importante comunidad de usuarios en los foros de Guadalinex

Figura 9: Logotipo de distribución Guadalinex

0.3.2 – Razones para el apoyo político al software libre.

El propósito de la Junta es dar a conocer el software libre por las ventajas y valores que representa.

El software libre no tiene un problema de partidarios. Es sorprendente numerosa la cantidad y actividad de los grupos de usuarios de software libre a lo largo de toda nuestra geografía. Por poner una medida real, si se reunen todos los desarrolladores de software libre de España, superarían a la mayor empresa de creación de software de este país.

Paradigma de esta situación a nivel internacional esta sourceforge.net. en este sitio Web hay más de 1.050.000 'desarrolladores' registrados y casi 100.000 proyectos arrancados (un proyecto normalmente representa una aplicación). Por poner un ejemplo es seguro que Microsoft no tiene ni 30.000 programadores entre sus casi 55.000 empleados.

Otro problema que no tiene el software libre es falta de apoyo político. Al menos, en España, hay un número razonable de políticos, tanto del PSOE, como del PP, de IU, de ERC, etc. que han usado el software libre como bandera y herramienta para la popularización de la sociedad de la información.

No solo seis comunidades autónomas han creado su propia distribución de software libre, sino que el resto han coqueteado en mayor o menor medida con el uso y/o promoción del software libre.

Finalmente tampoco es un problema de un monopolio preexistente en el mercado. Aunque es grande, desde luego no es el obstáculo definitivo a la difusión del software libre. Como ejemplo, el lenguaje Java ha pasado a ser casi el lenguaje número 1 del mercado desde el año 95 en que apareció en el mercado, y presenta pequeñas ventajas frente a sus competidores. El software libre presenta grandes ventajas y Linux empezó a crearse en el 91 y Richard Stallman está apoyando al software libre desde la FSF desde el año 85.

0.3.3 – Múltiples usos en nuestra comunidad

Existen varios "sabores" de Guadalinex, según a qué público esté orientada:

- **Guadalinex Base**, de propósito general, que se publica una vez al año.
- **Guadalinex EDU**: para los centros educativos.
- **Guadalinex CDM**: para los Centros de Día de Mayores.
- **Guadalinex BIB**: para Bibliotecas (actualmente en desarrollo).
- **Guadalinex Mini**: para ordenadores antiguos.
- **Guadalinex Guadalinfo**: para los centros Guadalinfo de la Junta de Andalucía

I - Unidad didáctica I. Introducción

I.I – Programación estructurada

La programación estructurada sigue tres reglas básicas de funcionamiento:

- **La secuencia**: indica que las instrucciones del código se leerán de principio a fin
- **La iteración**: la segunda indica que, según cierta condición, un número de instrucciones podrían repetirse un numero determinado de veces
- **La condición**: la tercera indica que según unas ciertas condiciones se ejecutarán o no un conjunto de instrucciones.

Veamos a continuación un algoritmo para limpiar platos en el que se ponen en práctica estas 3 reglas:

```
mientras haya platos
    coger plato
    mientras haya suciedad
      echar jabon
      pasar el estropajo por el plato
    si plato es azul
      ponerlo con los azules
```

En código no estructurado, quedaría algo más lioso:

```
1 coger plato
2 echar jabon
3 pasar el estropajo por el plato
4 si hay suciedad ir a la instrucción 2
5 si el plato no es azul ir a la instrucción 7
6 ponerlo con los azules
7 si hay más platos ir a la instrucción 1
```

Estructura de un programa

En la programación estructurada hay un inicio y un fin perfectamente bien definido de acuerdo al diagrama de flujo que se planteó al concebir la idea del programa.

Un programa bien estructurado debería tener algun subprograma que capture cualquier error dentro del programa principal o de cualquier subprograma dentro de la aplicación de tal modo que el subprograma que captura los errores genere un registro de datos que describa el error generado y/o en qué subprograma se generó el error para posteriormente corregirlo. Para facilitar la corrección de estos errores se hace uso de los comentario agregados en el código fuente.

Variables y constantes

Como hemos visto, el ordenador sigue una serie de instrucciones. Pero esas instrucciones tienen que operar sobre una serie de datos. El ordenador típico solo procesa una instrucción a la vez, por lo que necesita 'espacios de memoria' donde guardar o depositar, a modo de cajones, por usar un símil conocido, los diversos datos con los que trabaja. Aquí es donde entran en juego las variables y constantes.

Las **variables** son esos espacios de memoria en los que el ordenador puede leer o escribir datos. Por otro lado, habrá datos que no querremos modificar durante la ejecución del programa, por ser de naturaleza más invariable. Esto es lo se conoce como **constantes**, regiones de memoria con un valor fijo.

Comentarios

Son líneas de texto que el compilador o el intérprete no consideran como parte del código, con lo cual no están sujetas a restricciones de sintaxis y sirven para aclarar partes de código en posteriores lecturas y, en general, para anotar cualquier cosa que el programador considere oportuno.

Uno como programador debe tener como prioridad documentar nuestro código fuente ya que al momento de depurar nos ahorrará mucho tiempo de análisis para su corrección o estudio.

Los programadores profesionales tienen la buena costumbre de documentar sus programas con encabezados de texto(encabezados de comentarios) en donde describen la función que va a realizar dicho programa, la fecha de creación, el nombre del autor y en algunos casos las fechas de revisión y el nombre del revisor.

Por lo general algunos programas requieren hacer uso de llamadas a subprogramas dentro de una misma aplicación por lo que cada subprograma debería estar documentado, describiendo la función que realizan cada uno de estos subprogramas dentro de la aplicación.

Estructuras de datos y de control

a) Estructuras de control

Las estructuras de control pueden dividirse en dos: E**structuras de control Condicional** y **Estructuras de control Repetitivo**

Las **estructuras de control condicional** son las que incluyen alternativas de seleccion en base al resultado de una operación booleana, como por ejemplo, una comparación (A=B). Según la expresión sea cierta o falsa, se ejecutará un trozo de código u otro.

Ej.

```
IF A=0 THEN
      PRINT "A vale 0"
  ELSE
      PRINT "A no vale 0"
```

Otra sentencia de control son las de tipo **SWITCH CASE**. En este tipo de sentencias se especifica la variable a comparar y una lista de valores con lo que comparar. Aquel que sea el verdadero, se ejecutará:

```
SWITCH A
      CASE 0:
         PRINT "A vale 0"
      CASE 1:
         PRINT "A vale 1"
```

Otras herramientas imprescindibles del control de la ejecución de nuestro código son los BUCLES o CICLOS. Consisten en un método que permite repetir un trozo de código varias veces.

Existen 2 tipos de bucles:

- **Bucle FOR:** el bucle FOR consiste en una sentencia que engloba un grupo de instrucciones y tiene una variable cuyo valor se va modificando en cada vuelta

  Especificamos en este caso que A variará desde 0 hasta 10, con lo que repetiremos el bucle

  ```
  FOR A=0 TO 10
        PRINT "Estamos en el bucle"      10 veces.
  ```

  Con esto cerramos el bucle e indicamos el final del bloque de instrucciones que se repiten
  NEXT A

- **Bucle WHILE:** el bucle WHILE consiste en un bucle en el que el código se repite hasta que se cumpla alguna condición booleana (es decir, una expresión que de como resultado verdadero o falso). Hay variaciones, como el REPEAT...UNTIL, que se diferencia en el momento de comprobar si se hace verdadera o no la condición.

  Aquí especificamos la expresión que evaluamos y aquí se comprueba
  WHILE A<>(B*2) DO
      Incrementamos el valor de A hasta que sea igual a B*2
      A=A+1
  Como en el FOR, necesitamos especificar donde acaba el bucle y el codigo.
  DONE

1.2 – Ejercicios

Ejercicio 1

¿Qué devolvería el siguiente pseudocódigo si la variable **d** recibiera como valor de entrada el nº **5**?

```
SWITCH d
  CASE 1: PRINT "Lunes"
  CASE 2: PRINT "Martes"
  CASE 3: PRINT "Miércoles"
  CASE 4: PRINT "Jueves"
  CASE 5: PRINT "Viernes"
  CASE 6: PRINT "Sábado"
  CASE 7: PRINT "Domingo"
```

Ejercicio 2

¿Qué valor tendrá la variable **i** en la tercera iteración del siguiente bucle?

```
FOR i=-1 TO 5
      i = i + 1
NEXT i
```

Ejercicio 3

¿Qué resultado devolvería el siguiente pseudocódigo?

```
N=20
IF(N < 20)
      PRINT "El valor de N es menor a 20"
ELSE
      PRINT "El valor de N es igual o mayor a 20"
```

2 - Unidad didáctica 2. El entorno de programación Gambas

2.1 – Entorno Gambas

Las siglas Gambas quieren decir "Gambas Almost Means Basic", es decir, "Gambas casi quiere decir Basic". Gambas abre el entorno de la programación visual en Linux a todo el mundo, como lo hizo en su día Visual Basic en Windows.

La ampliación del lenguaje BASIC que alcanza Gambas amplia la potencia, profesionalidad y modernidad de las aplicaciones resultantes, sin abandonar la sencillez y claridad de este lenguaje de programación de alto nivel.

Gambas no es sólo un lenguaje de programación sino que constituye también un entorno visual de programación para desarrollo de aplicaciones visuales o en modo consola. Gambas está **orientado a eventos**, lo que significa que se realizan llamadas automáticas a procedimientos cuando el usuario de la aplicación elige un menú, hace clic con el ratón, mueve objetos en la pantalla, etc.

Podemos ver un aspecto del entorno de Gambas en la imagen siguiente (**ver Figura 1**)

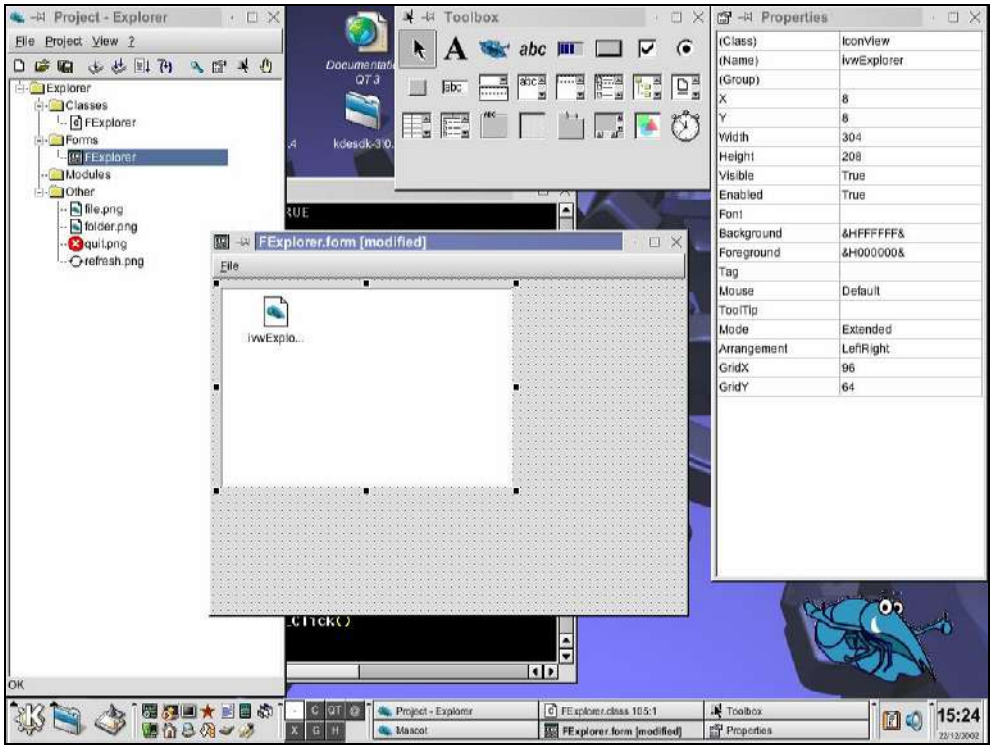


Figura 1: Entorno de desarrollo Gambas

A continuación se detalla una descripción de las ventanas más importantes que conforman dicho entorno:

Bienvenida

Pantalla de bienvenida al entorno Gambas (**ver Figura 2**). A través de esta ventana podemos, entre otras opciones:

- Crear un nuevo proyecto
- Abrir un proyecto existente
- Ver ejemplos
- Salir del programa
- Etc.

Figura 2: Inicio

Creación de proyecto

Desde esta ventana podemos elegir el tipo de proyecto a crear (**ver Figura 3**)

- De consola
- Basado en aplicación gráfica
- Generarlo mediante copia de un proyecto existente

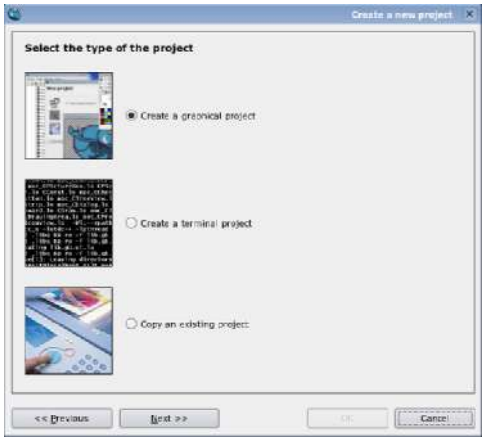


Figura 3: Selección del tipo de proyecto

Una vez elegido el tipo de proyecto a crear, el siguiente paso es configurar algunas propiedades básicas del mismo como por ejemplo el nombre del proyecto o el título para la aplicación (**ver Figura 4**)

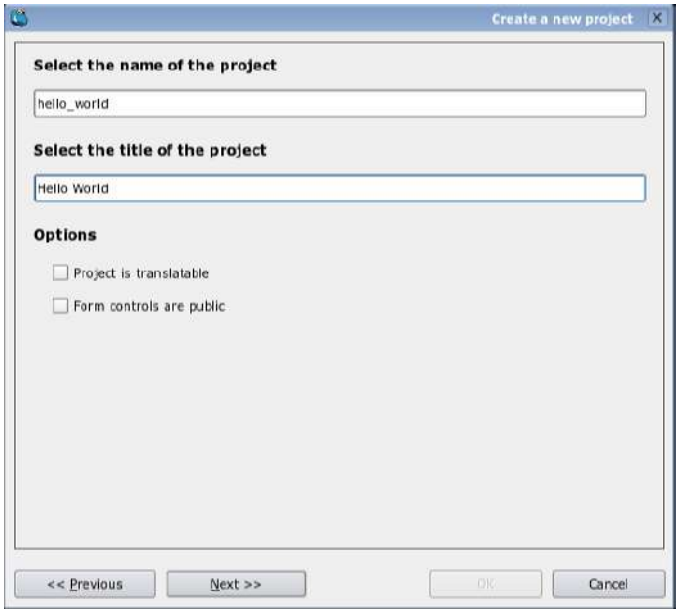


Figura 4: Configuración de las propiedades básicas del proyecto

Lo siguiente es establecer la carpeta en la que se ubicarán los ficheros del proyecto (**ver Figura 5**):

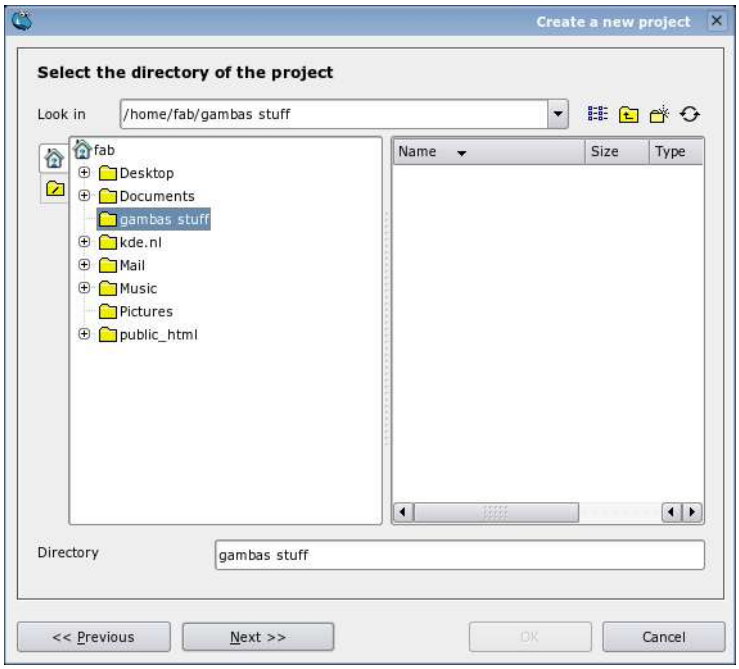


Figura 5: Selección de la carpeta para el proyecto

Barra de herramientas

En esta ventana se encuentran disponibles todas las herramientas que podemos emplear para construir nuestras aplicaciones de tipo gráfico (**ver Figura 6**)

Figura 6: Barra de herramientas

Ventana de propiedades

Esta ventana muestra las distintas propiedades de los componentes que vamos incluyendo en nuestra aplicación. Al seleccionar un componente (por ejemplo un botón), la ventana de propiedades se actualiza con las propiedades del componente seleccionado (**ver Figura 7**)

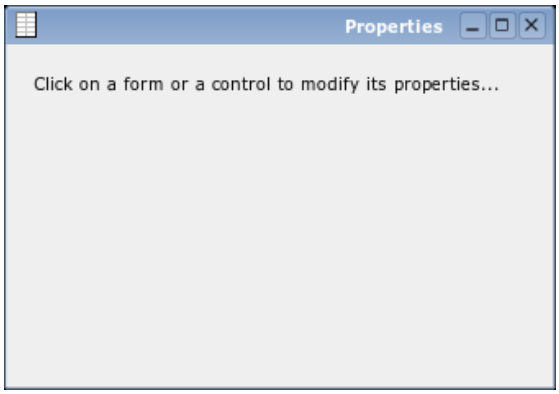


Figura 7: Ventana de propiedades (cambian las propiedades en función del formulario o control seleccionado)

Ventana de proyectos

Desde esta ventana podemos ver y gestionar los distintos proyectos que tenemos creados, así como sus ficheros y carpetas (**ver Figura 8**)

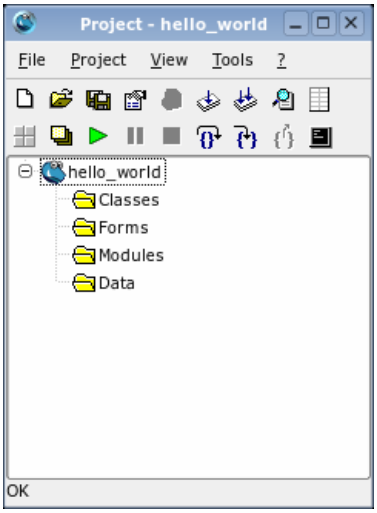


Figura 8: Ventana Gestor de proyectos

2.2 – Estructura de un programa en Gambas

Desde el inicio, Gambas nos ofrece 2 formas de enfocar la estructura de nuestros programas, incluso aquellos más simples como el típico "**Hola mundo!**":

- **Programación orientada a objetos:** paradigma típico de los lenguajes de programación más potentes
- **Programación estructurada**

En función de nuestra elección, el fichero que contenga el código del programa será una **clase** (programación orientada a objetos) o un **módulo** (programación estructurada).

Por simplicidad, de momento vamos a escoger la opción de programación estructurada, con lo que generaremos un archivo que contendrá el módulo correspondiente a nuestro primer programa en Gambas.

Para ello, haciendo clic con el botón derecho sobre el árbol de carpetas del entorno de Gambas aparecerá un menú contextual. Escogiendo la opción **Nuevo > Módulo** nos aparecerá una ventana en la que pondremos el nombre del módulo y a continuación pulsaremos el botón **Ok**, con lo que obtendremos nuestra primera ventana de programa, con el título indicado, en la que podremos escribir el código correspondiente.

El código del programa lo situaremos justo antes de la línea que contiene la palabra reservada **END.** En este caso, el código del programa se limita a imprimir en pantalla la cadena "Hola mundo" (**ver Figura 9**)

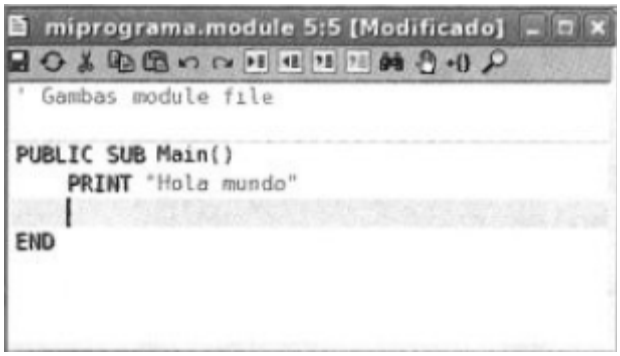


Figura 9: Primer programa en Gambas

Una vez escribimos el código, tras pulsar la tecla **INTRO** Gambas nos coloreará el texto de una forma particular. Los distintos colores posibles y su significado son los siguientes:

- **Gris**: aquellas líneas que comienzan con una comilla simple (’). Esto indica que la línea es un comentario
- **Azul**: palabras reservadas del lenguaje BASIC
- **Rosado**: cadenas de texto
- **Resalte amarillo**: aparece a la izquierda de aquellas líneas que han sido modificadas y que aún no han sido compiladas de nuevo

Con esto ya estaría listo el programa. Para comprobar su funcionamiento basta pulsar el botón verde con el símbolo de **Play**, que se encuentra en la pantalla del proyecto. Al hacerlo aparecerá una nueva ventana llamada **Consola** en la que se visualizará la salida del programa

2.3 – Sintaxis

Variables

Los programas almacenan la información de uso inmediato en memoria RAM. Trabajando al más bajo nivel, se deposita la información indicando la dirección de la memoria RAM directamente. Esto es muy engorroso, especialmente cuando el número de datos a almacenar es muy grande.

Para solucionarlo, los lenguajes de programación aportan el concepto de "variable": es nombre al cual nos referimos para almacenar información u obtenerla, de direcciones de memoria que se gestionan sin que a nosotros nos afecte directamente.

Nosotros indicaremos un nombre, por ejemplo "**MiValor**", asignaremos valores con el operador " **=** " (**MiValor = 3**), y haciendo referencia a ese nombre podremos leer su contenido (**PRINT MyValor** , mostraría en pantalla "**3**" en este ejemplo).

Según el lenguaje de programación se habla de que no está "tipado", o que está "tipado", en función del modo en que usa las variables. En un lenguaje "**no tipado**", cada variable puede almacenar cualquier cosa, (un número, una cadena de texto, etc). Esto puede estar bien para programar muy rápido, pero da lugar a graves errores en aplicaciones medianas y grandes. Gambas, por el contrario, es un lenguaje "**tipado**", y esto significa que de modo previo al uso de cualquier variable, forzosamente hemos de "declararla", especificando el tipo de datos que va a contener. Estos son algunos ejemplos de "declaraciones" con Gambas:

```
DIM MyVar As Integer
PRIVATE Respuesta As String
PUBLIC Suma As Float
```

En estos ejemplos, y sin que nos interese ahora la primera parte de estas declaraciones (**DIM**, **PRIVATE**, **PUBLIC**), estamos indicando que deseamos utilizar una variable llamada "MyVar" que almacenará números enteros, otra llamada "Respuesta" que almacenará cadenas de texto, y otra llamada "Suma" que almacenará números con coma flotante (es decir, es capaz de almacenar números con parte decimal)

Funciones

En los primeros tiempos de la informática, y en realidad, en la actualidad los microprocesadores siguen trabajando así, los programas eran una ristra

larguísima de ordenes, una detrás de otra, sin más posibilidad de "estructuración" que dar saltos hacia a delante o hacia atrás en el código.

Más adelante se impuso un modelo más "estructurado": dividir el código en bloques separados, llamados "funciones" o "métodos". Un programa completo estará formado por funciones, pequeños bloques de código a los que se le envían una serie de valores de los cuales se hace responsable para procesarlos, y retornar un valor, si procede.

Cada programa en Gambas empieza con una función "**Main**", o principal, y a lo largo de su código, esta función envía datos a otras funciones, y , si es necesario, recibe sus resultados. Este proceso se denomina "**llamada**", una función "llama" a otra función, esta puede "llamar" a otras, etc. Los valores que se envían para ser procesados, se denominan "parámetros".

Para indicar a Gambas el inicio de una función, emplearemos básicamente esta nomenclatura:

PUBLIC FUNCTION nombre_de_la_funcion (p1 As TipoDato,p2 As TipoDato...) As TipoDato

o bien

PUBLIC SUB nombre_de_la_funcion (p1 As TipoDato,p2 As TipoDato...)

Vamos a explicar a continuación las distintas partes de las declaraciones anteriores:

- La palabra **FUNCTION** significa que el código de la función va a retornar algún valor, mientras que **SUB** significa que se trata de un procedimiento, esto es, una función que no retornará ningun valor, (Ej. emitir un mensaje, pero sin retornar ningún valor)

- **nombre_de_funcion:** Es el nombre con que llamaremos a la función desde otras partes del código, y será un identificador único para ese fragmento de código en todo el programa.
- **(p1 As TipoDato, p2 As TipoDato...):** A continuación, se indican las variables que se pasarán como parámetros, separadas por comas, e indicando el tipo de la variable, del mismo modo que en las declaraciones antes explicadas. Según lo que vaya a hacer la función emplearemos más o menos parámetros.

- **As TipoDato:** Si hemos indicado la palabra clave "FUNCTION", indicaremos al final el tipo de dato que vamos a devolver. Dependiendo del tipo de parámetros que utilicemos, estos pueden tambien utilizarse en ocasiones para devolver el resultado deseado, al margen de este valor retornado.

Ámbito

No todas las funciones y variables se utilizan desde todo el código, hay varias razones para limitar las zonas desde donde se puede acceder a una variable o función: desde razones de rendimiento, por ejemplo, es mejor tener variables que

se creen y se destruyan tras ser usadas, que tener un "monstruo" en memoria con todas las variables permanentemente almacenadas, hasta razones de comodidad para el programador, ya que es mejor poder definir el mismo nombre para ciertas variables, en zonas "aisladas" del código, que tener que inventar miles de nombres a lo largo de un gran programa.

Gambas permite verias técnicas para separar o aislar el "**ámbito**" de las variables y funciones. Los programas escritos en Gambas, constan de "**módulos**", "**formularios**", y "**clases**". Pues bien, el código de cada módulo está parcialmente aislado del código de todos los demás, y de igual modo sucede con los formularios y clases. Es posible tener dos módulos que contengan una función con el mismo nombre, con la estructura siguiente:

Módulo 1:
Public SUB MiFuncion(Dato As Integer)
Módulo 2:
Public SUB MiFuncion(Dato As Integer)

Si nos encontramos dentro de otra función del módulo 1, yqueremos llamar a la función "MiFuncion", directamente lo indicaremos así en el código:

...
MiFuncion(34)
...

Al llamar a "MiFuncion", el intérprete sabe que nos referimos a la función que se encuentra dentro del mismo módulo. Ahora bien, si nos encontramos dentro del módulo 2, o de cualquier otro módulo, clase o formulario, hemos de especificar el módulo en el que se encuentra la función. Así, si desde el módulo 2 indicaramos:

...
Mifuncion(9654)
...

Realmente estaríamos llamando a la función con ese nombre que se encuentra dentro del módulo 2. Hemos de especificar que deseamos utilizar la del módulo 1, indicando el nombre del módulo, un punto, y el nombre de la función:

Modulo1.MiFuncion(9654)

La segunda técnica para delimitar el ámbito de la función, es indicar al intérprete si deseamos que esa función sea accesible desde otro módulo, formulario o clase. Si indicamos la palabra "PUBLIC", delante de la declaración de la función, esta podrá ser llamada desde otros módulos, clases o formularios, pero si indicamos "PRIVATE", sólo podremos acceder desde el propio módulo (se dice también que "no es visible" desde otros módulos)

En cuanto a las variables, éstas pueden ser declaradas al principio del módulo, formulario o clase, antes de cualquier función, en este caso emplearemos las palabras "**PRIVATE**" y "**PUBLIC**" de igual modo que con las funciones, y estas

variables existirán mientras exista el módulo, clase o formulario. Pero también podemos indicar una variable dentro de una función, empleando ahora la palabra clave "DIM", y en este caso la variable sólo existe desde que se entra en la función, hasta que se sale, y no es accesible en absoluto desde ninguna otra zona del código fuera de la función.

Ej.

```
PRIVATE Var1 As Integer
PUBLIC Var2 As String

PUBLIC FUNCTION SumaEspecial (V1 As Integer,V2 As Integer) As Integer
  DIM Media As Integer
  Media=(V1+V2)/2
  RETURN Media+V1+V2
END
```

2.4 – Ejercicios

Ejercicio 1

Crear un nuevo proyecto en Gambas, de nombre **MiPrimerProyecto**. Almacenarlo en una carpeta previamente creada con el mismo nombre.

Ejercicio 2

Sobre el proyecto anteriormente creado, crear un procedimiento que imprima el siguiente mensaje de bienvenida: "**Bienvenidos a mi primer proyecto en Gambas**".

Ejercicio 3

Sobre el proyecto creado en el ejercicio 1, construir una función que calcule si un número almacenado en una variable (**num**) es par o impar.

3 - Unidad didáctica 3. La caja de herramientas

3.1 – Componentes

El entorno de Gambas incorpora una serie de componentes o herramientas que nos van a permitir construir nuestras aplicaciones gráficas. Estos componentes se encuentran agrupados en la barra de herramientas de Gambas:

Figura 1: Barra de herramientas en Gambas

Algunos de los controles más representativos de esta barra de herramientas son:

- **Herramienta de selección (Selection):** el único elemento de la caja de herramientas que no es realmente un control. La herramienta de seleccion proporciona un puntero negro de ratón que nos permite seleccionar y trabajar con formularios y sus controles

Figura 2: Herramienta Puntero

- **Etiqueta (TextLabel):** permite introducir elementos de texto en los formularios.

Figura 3: Herramienta Etiqueta

- **Campo de texto (TextBox):** este control sirve para insertar cajas de texto en nuestros formularios, a través de los cuales el usuario pueda introducir información en la aplicación

Figura 4: Herramienta Cuadro de texto

- **Control Imagen (Image):** permite insertar imágenes en los formularios

Figura 5: Herramienta Imagen

- **Control Barra de progreso (ProgressBar):** barra de progeso que nos muestra de forma gráfica el porcentaje de completitud de una determinada tarea o proceso

Figura 6: Herramienta Barra de progreso

- **Botón (Button):** botón de acción que puede ser pulsado para llevar a cabo una determinada acción

Figura 7: Herramienta Botón de Opción

- **Cuadro de chequeo (Checkbox):** casilla de verificación, admite 2 estados (chequeado o no chequeado), que cambian al pulsar sobre ella con el ratón.

Figura 8: Herramienta Cuadro de chequeo

- **Botón de opción (Option Button):** parecida a la anterior, la diferencia es que en un grupo de botones de opción, sólo uno de ellos puede estar seleccionado al mismo tiempo (en un grupo de cuadros de chequeo, puede haber más de uno)

Figura 9: Herramienta Botón de opción

- **Cuadro combinado (ComboBox):** lista de elementos plegada que despliega sus opciones al ser pulsada con el ratón

Figura 10: Herramienta Cuadro combinado

- **Cuadro de lista (ListBox):** parecido al anterior pero con sus opciones desplegadas.

Figura 11: Herramienta Cuadro de lista

- **Control area de texto:** cuadro de texto de grandes dimensiones para la introducción de grandes cantidades de texto. Incorpora barras de desplazamiento automáticas para desplazarnos por su contenido, en caso de que este supere sus dimensiones.

Figura 12: Herramienta Area de texto

- **Control Marco (Frame):** permite insertar un marco para la agrupación de varios elementos o controles.

Figura 13: Herramienta Marco

- **Control Pestañas (TabStrip):** control que permite agrupar otros controles en un conjunto de pestañas. Al cambiar de pestaña se visualizan el conjunto de controles incluidos en la petaña seleccionada.

Figura 14: Herramienta Pestañas

- **Control Vista de árbol (TreeView):** este control permite incluir un control en el que sus elementos (conjunto de carpetas y ficheros) se organizan en forma de árbol: las carpetas constituyen las ramas, y los ficheros incluidos en ellas las hojas. Mediante el ratón es posible plegar y desplegar cada una de las carpetas (ramas) del árbol, mostrando/ocultando los ficheros contenidos en ellas (hojas).

Figura 15: Herramienta Vista de árbol

- **Control vista con scroll (ScrollView):** control que permite insertar un área para mostrar contenido incluyendo barras de desplazamiento.

Figura 16: Herramienta Vista de árbol

- **Control Temporizador (Timer):** control que permite llevar a cabo ciertos procesos o eventos de forma periódica. El usuario debe fijar la duración del intervalo.

Figura 17: Herramienta Temporizador

- **Control Area de dibujo (DrawingArea):** control que permite insertar en el formulario un área para poder dibujar formas y contornos gráficos.

Figura 18: Herramienta Area de dibujo

3.2 – Ejercicios

Ejercicio 1

Crear un proyecto en Gambas llamado **Calculadora**, que mediante 4 botones (representando a las distintas operaciones matemáticas elementales: suma, resta, multiplicación y división) y 3 cuadros de texto (operando 1, operando 2 y resultado), permita al usuario realizar los cálculos necesarios, a modo de calculadora electrónica.

Ejercicio 2

Crear un proyecto en Gambas llamado **Curriculum** que muestre, mediante un control de pestañas y los controles apropiados, las distintas secciones de un currículum vitae: **datos personales**, **titulación**, **experiencia laboral**, **formación** y **otros datos de interés**. Cada una de estas secciones debe ubicarse en una pestaña diferente.

Ejercicio 3

Hacer un proyecto en Gambas llamado **Reloj** que muestre en un formulario la hora actual en una etiqueta. Se deben mostrar el valor de hora, minutos y segundos, de forma que la hora se actualice tras cada segundo.

4 - Unidad didáctica 4. Gestión de eventos

4.1 – Tipos de eventos

La interfaz gráfica ofrece una serie de eventos para modelar el comportamiento de un programa, es decir, el conjunto de acciones a ejecutar en el momento en que ocurra uno de dichos eventos.

Los eventos se asocian con los distintos componentes de la barra de herramientas que podemos incluir en nuestras aplicaciones. De esta forma, tenemos por ejemplo, eventos para los cuadros de texto (Ej. acciones a ejecutar al introducir o modificar el contenido del cuadro de texto), eventos de los botones (Ej. acciones a ejecutar al pulsar con el ratón sobre el botón), eventos para los cuadros combinados (Ej. acciones a ejecutar al desplegar la lista de elementos del cuadro combinado), etc.

Existe una gran diversidad de eventos que podemos controlar. Los más relevantes son:

- **Pulsación de teclas en el teclado**: tiene lugar cuando el usuario pulsa una tecla o conjunto de teclas del teclado
- **Eventos del ratón (pulsación de botón izquierdo o derecho, soltar botón, arrastrar):** las acciones que el usuario puede llevar a cabo por medio del ratón también son controlables por eventos.
- **Operaciones con ventanas de formulario (abrir, cerrar, minimizar, maximizar, redimensionar)**

4.2 – Gestión de eventos

Además de los eventos predefinidos en Gambas, podemos crear nuestros propios eventos personalizados, y no sólo en programas gráficos, si no también en programas de consola, ya que la gestión de eventos de Gambas es independiente del entorno gráfico, a diferencia de otros lenguajes.

Para explicar mejor la forma de gestionar y personalizar nuestros propios eventos, vamos a verlo con un ejemplo paso a paso:

1. Para empezar, crearemos un módulo de clase llamado, en nuestro ejemplo, **ClsElemento**. Esta clase de ejemplo, tendrá un sólo método **Llamada()**, que incrementa un contador. Una vez que el contador alcanza el valor **10**, se resetea el contador, y se dispara un evento para informar de ello.
2. Lo primero es declarar el evento. En nuestro caso, enviará como parámerto una cadena advirtiendo que el contador vale 10:

   EVENT MiEvento(Data AS String)

3. Tras esto, añadimos una variable privada que actúa como contador:

```
PRIVATE Contador AS Integer
```

4. Escribimos ahora el código de la función de llamada:

```
PUBLIC SUB Llamada()

  ' Incrementamos el contador
  Contador=Contador+1
  IF Contador>9 THEN
    ' Si el valor del contador es 10
    ' disparamos el evento
    RAISE MiEvento("Contador vale 10")
    ' Y reseteamos el contador
    Contador=0

  END IF
END
```

5. Finalmente, escribimos el código principal del programa. Para ello cremos un formulario con un botón. Al inicio declaramos un objeto de nuestra clase:

```
PRIVATE MiClase AS ClsElemento
```

Al abrirse el formulario, creamos el objeto, declarando el "observador" de eventos entre comillas:

```
PUBLIC SUB Form_Open()
MiClase=NEW ClsElemento AS "MiObservador"
END
```

Añadimos al botón el código para que utilize el objeto:

```
PUBLIC SUB Button1_Click()

  MiClase.Llamada ()

END
```

Y finalmente, recogemos el evento:

```
PUBLIC SUB MiObservador_MiEvento(Data AS String)

  Message.Info (Data)

END
```

Esta función tiene el nombre del **observador** de eventos, un guión bajo, y el nombre del evento, por lo que será llamada cuando se dispare el evento:

Código completo de "ClsElemento":

```
' Gambas class file
```

```
PRIVATE Contador AS Integer
EVENT MiEvento(Data AS String)

PUBLIC SUB Llamada()

  Contador=Contador+1
  IF Contador>9 THEN

    RAISE MiEvento("Contador vale 10")

    Contador=0

  END IF

END
```

Código completo de "Form1":

```
PRIVATE MiClase AS ClsElemento

PUBLIC SUB Button1_Click()

  MiClase.Llamada ()

END
PUBLIC SUB MiObservador_MiEvento(Data AS String)

  Message.Info (Data)

END


PUBLIC SUB Form_Open()

  MiClase=NEW ClsElemento AS "MiObservador"

END
```

4.2 – Ejercicios

Ejercicio 1

Crear un proyecto en Gambas llamado **Saludo**, que contenga un formulario y un botón con el texto "**Pulsa aquí**". Dicho botón al ser pulsado debe mostrar un mensaje en una etiqueta que ponga "**Hola, bienvenido a la programación en Gambas**".

Ejercicio 2

Modificar el proyecto **Saludo**, creado en el ejercicio anterior, para que muestre el saludo cuando se pulse también la tecla "**s**" del teclado.

Ejercicio 3

Crear un nuevo proyecto en Gambas llamado **Listas**, que incluya un formulario, un control lista de elementos, un cuadro de texto y 2 botones, con los textos **Añadir** y **Borrar**. El funcionamiento de la aplicación será el siguiente:

- El botón **Añadir** añade el texto introducido en el cuadro de texto a la lista al ser pulsado.
- El botón **Borrar** elimina el elemento de la lista seleccionado. Si no hay ningún elemento de la lista seleccionado el botón debe permacener en un estado inactivo, al igual que si no hay elementos en la lista. Dicho botón se activará cuando el usuario seleccione algún elemento de la lista.

5 - Unidad didáctica 5. Diseño de formularios

5.1 – Creación de formularios

Con Gambas se pueden hacer aplicaciones o programas con interfaz gráfica de forma muy rápida, pues integran un diseñador de formularios o ventanas muy potente.

Una vez creado un proyecto (ver unidades didácticas anteriores), para crear un formulario tan sólo tenemos que pulsar sobre el apartado **Formularios** y seleccionamos la opción **Nuevo formulario**, apareciéndonos la nueva ventana correspondiente al formulario creado (**ver Figura 1**)

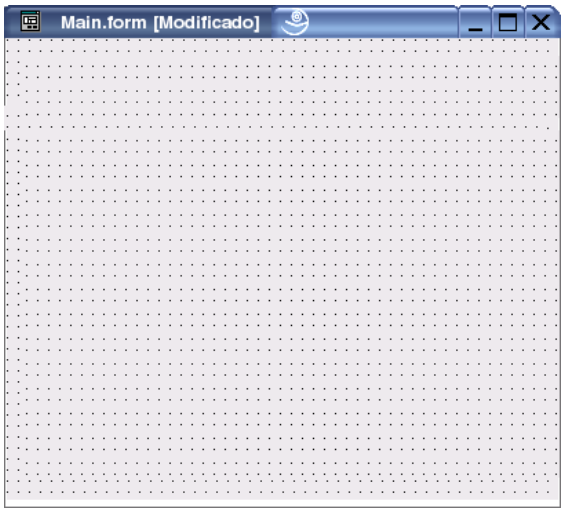


Figura 1: Nuevo formulario

5.2 – Diseño de formularios

Una vez creado un formulario, el siguiente paso es darle un diseño o formato, en base a los componentes (controles) necesarios a insertar en para desarrollar nuestra aplicación.

Estos componentes están disponibles en la barra de controles de Gambas (**ver Figura 2**). El funcionamiento de cada uno de ellos fue tratado en la unidad didáctica 3.

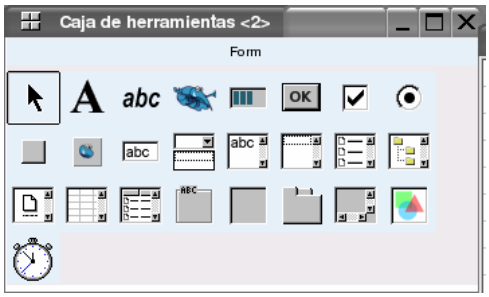


Figura 2: Controles de la barra de herramientas de Gambas

Vamos a continución a diseñar un formulario de ejemplo, en el que vamos a incluir los siguientes controles:

- Etiquetas (control **Label**)
- Lista de elementos (control **ListBox**)
- Botones (control **Button**)
- Panel (control **Panel**)
- Menú de opciones (control **Menu**)

Para incluirlos, basta seleccionar el control en cuestión de la barra de controles y a continuación "dibujarlos" sobre el formulario. A la hora de dibujarlos con el ratón podemos darle las dimensiones que deseemos (altura, anchura). Posteriomente, podemos editar las características de cada control a partir de la paleta de propiedades (color, texto asociado, tipo de letra, etc.)

El aspecto del formulario, ya con sus controles creados, sería el siguiente (**ver Figura 3**)

Figura 3: Diseño del formulario

Destacar el hecho de que los botones **Abrir**, **Guardar** y **Salir** no han sido dibujados directamente sobre el formulario, sino que están contenidos dentro del control **Panel**. Esta es una forma sencilla de agrupar distintos controles o elementos dentro de una interfaz.

Comentar también que en el texto incluido en los botones, existe una letra que contiene un carácter de subrayado. Eso indica que dicha letra sirve para acceder a la pulsación del botón a través de un **atajo de teclado** (combinación de teclas).

Para conseguir que los botones respondan a atajos de teclado, hay que poner un **amperstand** (**&**) delante de la letra que servirá como atajo.

Por otro lado, en la parte superior del formulario se ha situado un menú de opciones. Para su creación, pulsamos con el botón derecho en cualquier punto vacío del formulario y seleccionamos la opción **Editor de menú**, con lo que desplegaremos una ventana que nos permite configurar la estructura y opciones del menú **(ver Figura 4**)

Figura 4: Editor de menús

Dicha ventana nos permite configurar las propiedades de los distintos elementos del menú (nombre, título, atajo, etc). De entre ellas, destacar la opción **Grupo**, pues si tenemos varios controles (p. ej., el menú Abrir y el botón Abrir) que deben hacer lo mismo, asociándolos al mismo grupo sólo tenemos que escribir el código correspondiente al grupo de acciones al que pertenece cada control.

Así pues, en nuestro programa de ejemplo, hemos asociado al grupo "Abrir el menú" y el botón "Abrir", al grupo "Guardar" el botón y el menú "Guardar", etc

Tan sólo nos queda configurar como se va a comportar nuestra aplicación al emplear dichos controles (pulsación de un botón, adición de nuevos elementos a la lista, selección de una opción del menú, etc.)

Si ahora hacemos click en un botón o en el menú correspondiente, se nos abrirá el editor de código posicionándose el cursor en la declaración de un procedimiento que se llama igual que el grupo de acciones. Aquí deberemos introducir las acciones a llevar a cabo al pulsar el botón u opción de menú (**ver Figura 5**)

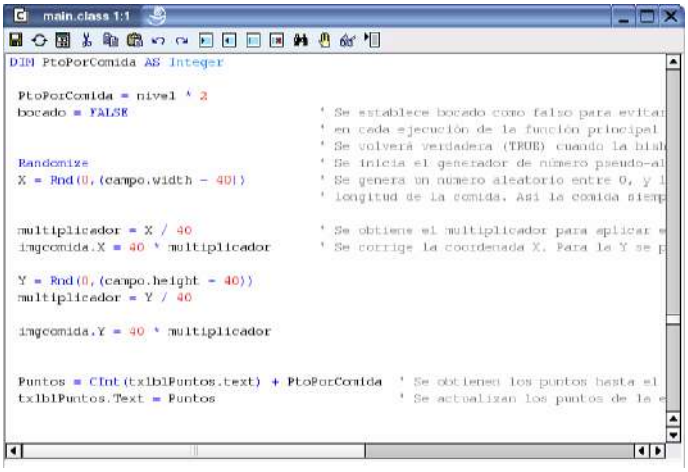


Figura 5: Editor de código (programación de eventos asociados a los controles)

5.3 – Consideraciones relativas al diseño

A la hora de crear nuestras aplicaciones en Gambas a través de formularios, debemos considerar una serie de cuestiones:

- No todos los usuarios utilizan la misma resolución de pantalla, gestor de ventanas y tipo de fuentes. Hay que tener cuidado y no tratar de "aprovechar" demasiado el espacio. Podemos acabar con etiquetas de texto (Label) ilegibles, botones con el texto cortado, etc
- Por la misma razón, conviene que la ventana principal de la aplicación sea redimensionable por el usuario (en Gambas es la propiedad **Border** del formulario. No es recomendable fijar esta propiedad a **Fixed**.
- Al crear el formulario, se nos ofrecen varias opciones que pueden ser interesantes (**ver Figura 6**):

Figura 5: Opciones de formulario

Las opciones relativas al constructor y destructor nos sirven en el caso de que queramos hacer alguna operación sobre el formulario antes de visualizarlo y al cerrarlo, respectivamente.

Aparecen las siguientes declaraciones:

' Gambas class file PUBLIC SUB _new() END PUBLIC SUB _free() END PUBLIC SUB Form_Open() END

Si seleccionamos elegimos las opciones "Constructor estático" y "Destructor estático", las declaraciones que nos aparecen ahora en el editor de código son:

'Gambas class file STATIC PUBLIC SUB _init() END STATIC PUBLIC SUB

_exit() END PUBLIC SUB _new() END PUBLIC SUB _free() END PUBLIC SUB

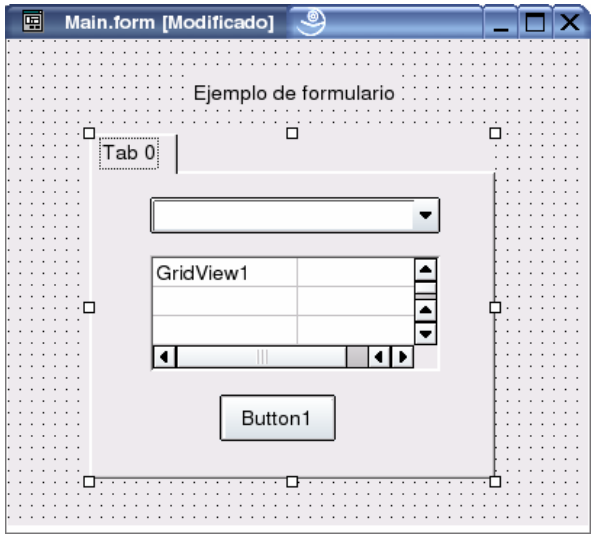
Form_Open() END

Podemos así alterar el comportamiento de nuestra aplicación al abrirse y/o cerrarse el formulario. Que el procedimiento esté declarado como STATIC significa que sólo podrá acceder a variables declaradas también como STATIC.

5.4 – Ejercicios

Ejercicio 1

Crear un nuevo proyecto, y asignarle el nombre **miformulario**. A continuación, diseñar un formulario que contenga los siguientes controles:



Ejercicio 2

Modificar las siguientes propiedades de los controles del formulario creado en el ejercicio anterior:

- Texto asociado al botón: **Aceptar**
- Texto asociado al control pestañas: **Alumnos**
- Texto para la etiqueta: **Listado de alumnos**
- Color de fondo del formulario: **Blanco**
- Modificar la anchura del botón aproximadamente al doble de la actual y a continuacion cambiar el texto a **Guardar cambios realizados**

Ejercicio 3

Incluir un menú en la parte superior del formulario. Dicho menú debe contener las siguientes opciones:

- **Archivo:** con los submenús **Nuevo, Abrir, Cerrar** y **Salir**
- **Edición:** con los submenús **Copiar** (atajo de teclado en la tecla **p**), **Cortar** (atajo de teclado en la tecla **t**), **Pegar** (atajo de teclado en la tecla **g**) y **Deshacer** (atajo de teclado en la tecla **d**).
- **Salir:** programar el evento **Click** (pulsación del botón) para hacer que el programa finalice al pulsar el botón del formulario. Esta misma acción debe ser posible llevarla a cabo desde la opción **Salir** del menú superior.

6 - Unidad didáctica 6. Aplicaciones multi-idioma

6.1 – Aplicaciones multi-idioma

Gambas incorpora soporte nativo para crear aplicaciones que se distribuyan entre usuarios que hablan idiomas distintos. Un proyecto ha de ser marcado como "Traducible" para que esto sea posible.

Tenemos 2 situaciones posibles a la hora de abordar la traducción de aplicaciones en Gambas:

1. **Si vamos a crear una nueva aplicación**

   En tal caso, creamos un nuevo proyecto gráfico con Gambas. En el cuadro de diálogo dónde se solicita el nombre del proyecto, tenemos un cuadro de chequeo disponible para establecer el nuevo proyecto como traducible. Marcamos dicha opción (**ver Figura 1**)

Figura 1: Configurando un nuevo proyecto multi-idioma

   El proyecto gestionará de forma automática los textos a partir de ese momento, de forma que estén disponibles en varios idiomas.

2. **Si la aplicación ya está creada**

   En este caso los pasos a seguir son:

   a) Abrir el menú **Proyecto** y dentro de este pulsar la opción **Propiedades**.

b) En el asistente de propiedades del proyecto, ir a la pestaña **Traducción** y marcar la opción **El proyecto es traducible**. Aceptar los cambios (**ver Figura 2**)

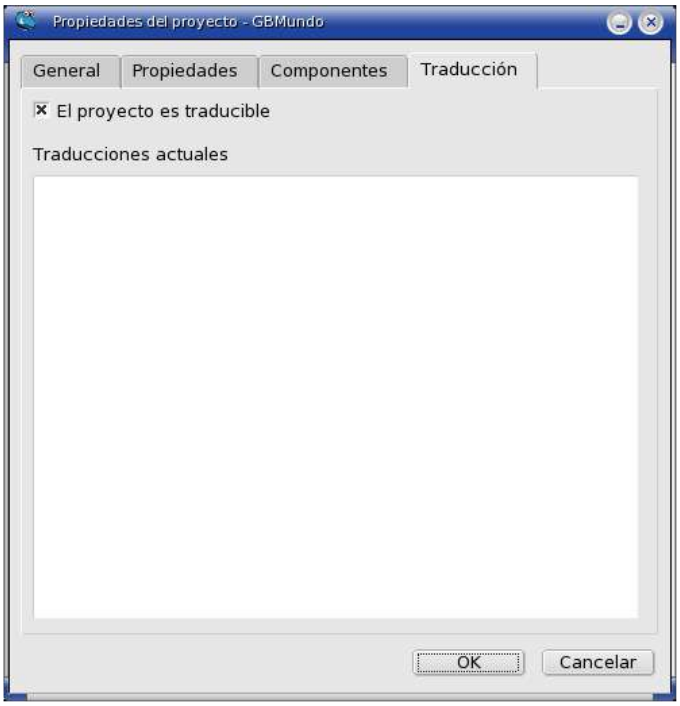


Figura 2: Configurando una aplicación existente como multi-idioma

Una vez establecida la opción multi-idioma, en cualquiera de los casos anteriores, todos los textos de la interfaz gráfica, se gestionan de forma automática, de modo que al acudir al **gestor de traducciones**, los encontraremos disponibles tras cualquier cambio o adición en la intefaz.

En cuanto a los textos del programa, hemos de indicar explícitamente que deseamos tenerlos disponibles para su traducción, ya que podría tratarse de textos que no se deben traducir, como, por ejemplo, cadenas SQL, o etiquetas de XML. Si en un código no traducible, indicamos las cadenas como en este ejemplo:

```
PUBLIC SUB Button1_Click()

   Message.Info ("Hola Mundo")

END
```

En el caso de un texto ha traducir, hemos de delimitar la cadena textual con signos de paréntesis:

```
PUBLIC SUB Button1_Click()

   Message.Info ( ("Hola Mundo") )

END
```

El compilador sabe distinguir automáticamente entre los paráentesis que corresponden al método, y los referidos a la cadena

Por último, ten en cuenta que hay signos especiales o "de escape" en Gambas; así, por ejemplo, el símbolo " & " se utiliza para indicar en la interfaz gráfica, que esa tecla será el atajo de teclado para acceder al menú o botón, por ejemplo. Otros símbolos tienen significado similar al que tienen en C o algunas shells, como es el caso de "\n" (retorno de carro). el gestor de traducciones, es capaz de comprobar la coherencia de caracteres especiales escritos en la versión original, y en la traducción.

El gestor de traducciones

El gestor de traducciones es una herramienta incluida en el entorno de Gambas que nos va a permitir establecer como Gambas va a gestionar las traducciones a la hora de diseñar aplicaciones multi-idioma.

Para acceder a él, abrimos el proyecto en cuestión que vamos a configurar como multi-idioma, y una vez abierto, acudimos al menú **Proyecto**, seleccionando la opción **Traducir** (**ver Figura 3**)

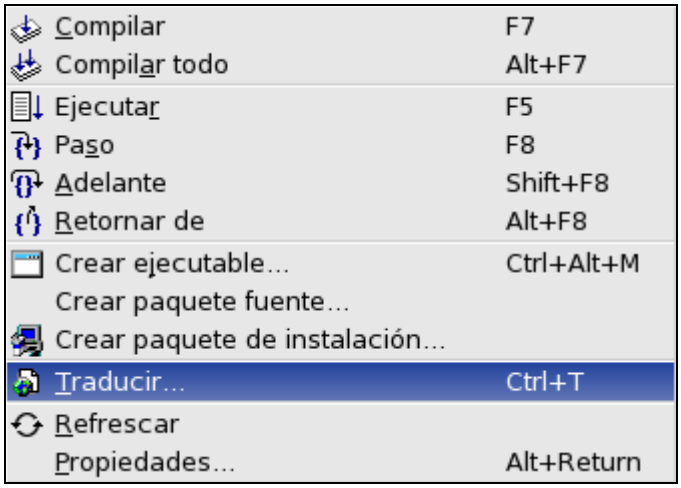


Figura 3: Opción **Traducir** (acceso al Gestor de Traducciones)

Una vez seleccionada la opción, entramos en el asistente de traducción. Hemos de elegir el idioma que deseamos traducir. Si ya habíamos trabajado previamente en ella, aparecerán los textos sin traducir, así como los ya traducidos. Todas las cadenas sin traducir, así como las introducidas con posterioridad a la última fase de traducciones, aparecerán con la traducción en blanco (**ver Figura 4**)

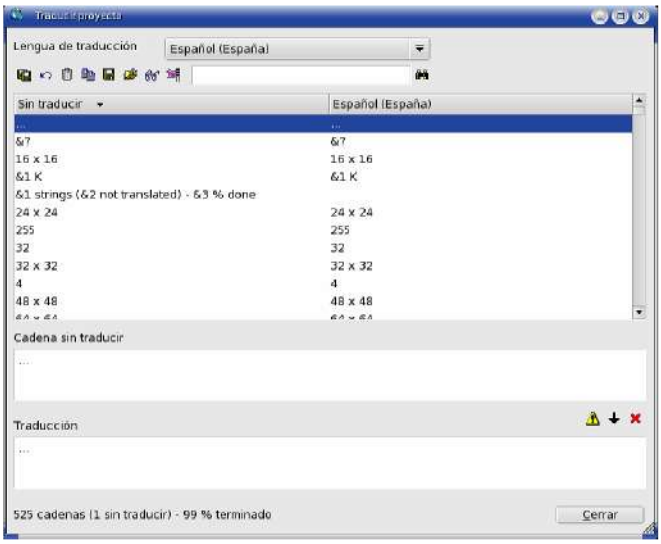


Figura 4: Gestor de Traducciones de Gambas

Cada vez que pulsamos con el botón izquierdo en una de las cadenas en la primera lista, en las listas inferiores se nos muestra la cadena original, y en la parte inferior disponemos de la caja de texto para escribir la traducción correspondiente. Cuando cambiamos a otra cadena, la que habíamos editado queda guardada durante esta sesión (es necesario pulsar **Guardar traducción** para que quede guardado permanentemente) (**ver Figura 5**)

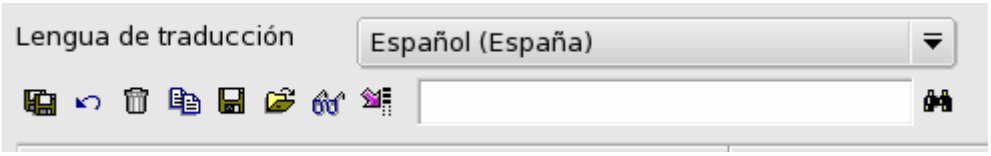


Figura 5: Gestionando traducciones

El resto de las opciones son las siguientes:

- **Guardar traducción**: los cambios realizados en esta sesión, se guardan y quedan disponibles para los usuarios del programa en ese idioma.
- **Recargar traducción**: se borran todos los cambios realizados en la última sesión, y se vuelve al estado anterior.
- **Borrar traducción**: elimina todos los textos traducidos.
- **Duplicar una traducción**: hace una copia de la traducción que había en otro idioma, de modo que nos sirva como base para el idioma a traducir. Puede ser útil, por ejemplo, para hacer una traducción en "Español (variante Argentina)", basada en la original "Español (variante España)". Lo mismo ocurre con el Francés de Bélgica o Canadá respecto al de Francia, que puede llegar a ser confuso según la nacionalidad del usuario, si no se realizan adaptaciones.
- **Exportar una traducción**: convierte la traducción en un fichero estándar tipo ".po", usado por la mayoría de los programas escritos en C y C++ sobre GNU/Linux, FreeBSD y software libre para otros sistemas operativos. este

formato permite enviar la traducción realizada a otras personas, en un sistema de desarrollo donde se trabaje en diferentes lugares. Este es el caso de Gambas: una vez escrita la traducción, se exporta a formato .po, y se envía a la lista de desarrollo de Gambas.

- **Importar una traducción**: proceso inverso al anterior. El coordinador recibe un fichero ".po", y lo absorve para incorporarlo a la versión de programa.
- **Verificar traducción**: imprescindible antes de dar por buena una nueva versión, comprueba que todos los símbolos especiales (por ejemplo '&') coinciden entre los textos originales y los traducidos. No debe utilizarse, ni exportarse nunca, una traducción en la cual el verificador da un mensaje de error.
- **Buscar la siguiente cadena no traducida**: nos dirige directamente a la siguiente cadena pendiente de traducir desde la anterior traducción, de modo que sea más fácil traducir conforme se incorporan nuevas cadenas al programa

3.2 – Ejercicios

Ejercicio

Elegir uno de los proyectos realizados en Gambas en unidades anteriores y convertirlos en una aplicación multi-idioma.