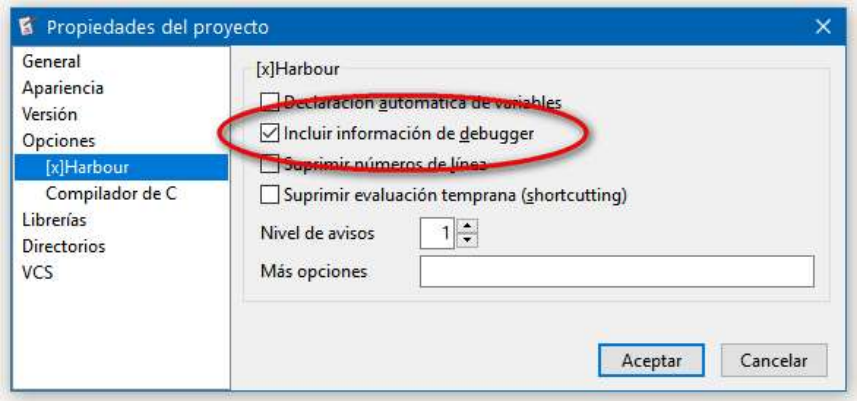
Depurar programas Harbour con XEdit

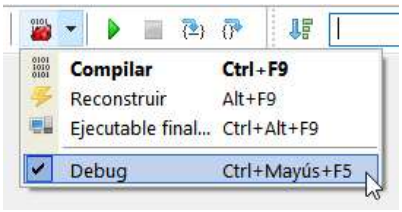
XailerDebug es una librería que habilita el uso del debugger integrado en XEdit con cualquier programa escrito en Harbour.

Para utilizar el debugger hay que compilar el programa con el flag **/B** o activar el modo debug en el proyecto XEdit. Se puede activar de tres formas:

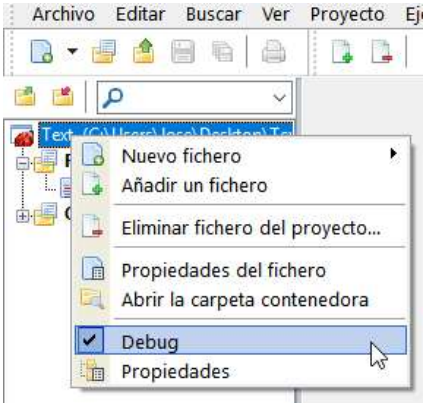
- Desde las propiedades del proyecto



- Activando la opción correspondiente al desplegar el botón "compilar" de la barra de botones



- Activándolo desde el menú contextual del proyecto en el gestor de proyectos



Además de activar el debugger, también hay que enlazar la librería XailerDebug con el programa. Esta librería está en dos formatos, y hay que enlazar la que corresponda al compilador de C que estemos utilizando. XailerDebug.lib es para BCC++ y libXailerDebug.a es para MinGW. La librería para BCC++ ha sido compilada con BCC++ 5.5 pero seguramente funcionará con versiones posteriores de BCC++ en 32 bits. Por otro lado, la librería para MinGW ha sido compilada con GCC 7.3, y al igual que la anterior es muy probable que funcione con otras versiones de MinGW, tanto anteriores como posteriores, ya que no tiene dependencias fuera de lo común.

Una vez compilado y enlazado nuestro programa con el debugger, podemos ejecutarlo desde XEdit y depurarlo con toda facilidad.

Puntos de ruptura

Se puede crear un punto de ruptura (*breakpoint*) en cualquier línea de código del programa, excepto en aquellas líneas que no son ejecutables, como líneas en blanco, declaraciones, comandos del preprocesador, etc.. En el caso de líneas de código partidas en varias líneas de texto (terminadas con el carácter ;) debemos establecer el punto de ruptura en la última de ellas.

Para crear un punto de ruptura hay que pulsar ***Ctrl*+*F5*** en la línea de código que queramos, o hacer clic con el ratón en el margen izquierdo del editor mientras pulsamos la tecla ***Ctrl***. Las líneas marcadas con un punto de ruptura aparecen con el fondo en color rojo claro en el editor.

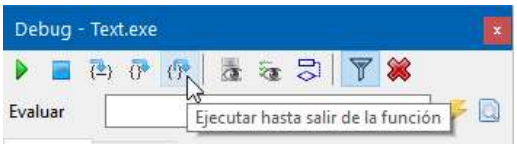
Se pueden establecer un máximo de 32 puntos de ruptura.

Además de los puntos de ruptura, podemos poner en nuestro código una llamada a la función *Altd()*, que invocará al debugger y se detendrá en la línea siguiente.

Ejecución paso a paso

Podemos ejecutar nuestro programa paso a paso pulsando las teclas ***F7*** o ***F8***. Ambas ejecutan una línea de código, pero si dicha línea de código es una llamada a otra función, la tecla ***F8*** ejecuta dicha función completamente y se detiene en la línea siguiente a la llamada. Por el contrario, la tecla ***F7*** salta dentro del código de la función que se ha llamado y la ejecuta también paso a paso.

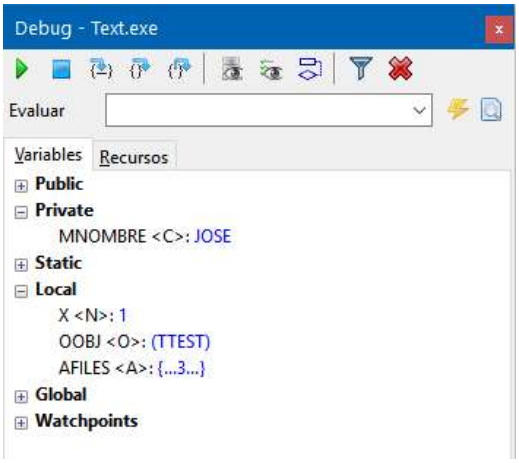
En la ventana del debugger hay también un botón que nos permite *ejecutar hasta salir de la función*. Lo que hace es continuar por sí mismo la ejecución de la función en curso y se detiene en la línea de código justamente posterior a la llamada a esta función.



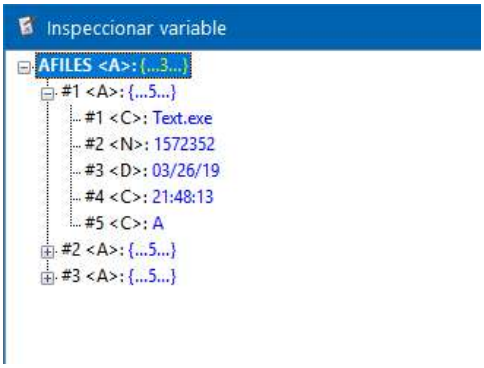
La ejecución paso a paso se puede lanzar desde antes de ejecutar el programa. Es decir, si antes de ejecutar el programa pulsamos ***F7*** o ***F8***, se ejecutará el programa y se detendrá en la primera línea de código ejecutable.

Inspección de variables

Cuando la ejecución del programa esté detenida por el debugger podremos ver las variables que tengamos en ámbito en la ventana del depurador. En esta ventana aparecen separadas por categorías (*Public*, *Private*, *Static* y *Local*). El grupo *Global* solo tenía sentido en xHarbour, ya que era un tipo de variables que solo soportaba este compilador, y por lo tanto en Harbour no tiene ningún uso.

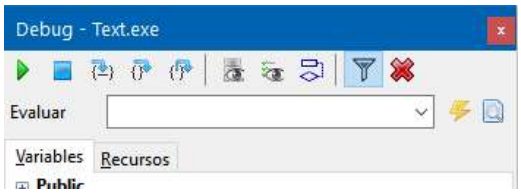


Todos estos grupos aparecen inicialmente cerrados. Para ver las variables que correspondan a un tipo determinado tendremos que abrir dicho grupo, y podemos ver el nombre, el tipo y el valor de cada una de ellas. Los valores de tipo complejo (arrays, objetos, hashes,...) no se muestran directamente. Para ver su contenido tendremos que hacer doble clic sobre esa variable y se abrirá una nueva ventana donde podremos inspeccionarla al completo.



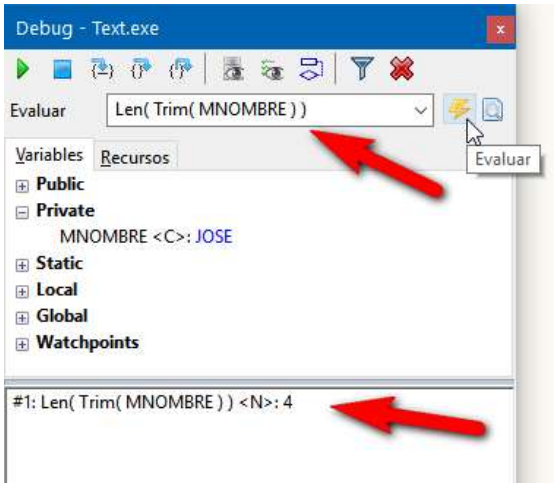
Evaluar una expresión

En la ventana del debugger hay un control donde podemos escribir cualquier expresión a evaluar.

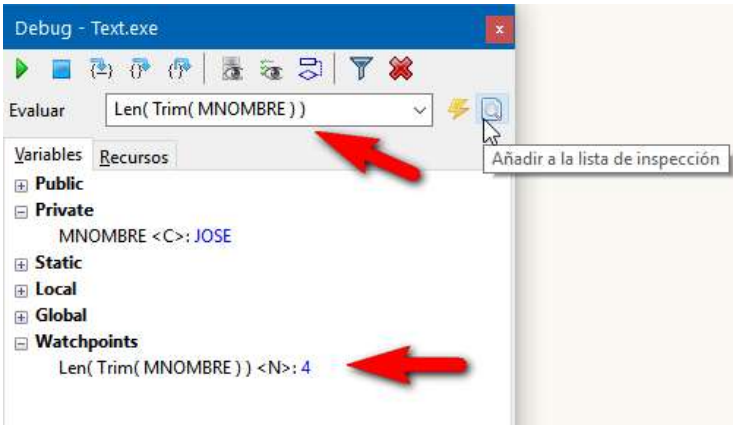


Esta expresión deberá ser computable en el ámbito de ejecución donde esté detenido el debugger. P.ej, si queremos evaluar una expresión que contiene una variable de tipo local, esa variable deberá estar accesible desde la función donde esté detenido el programa.

Una vez escrita la expresión, podemos pulsar ***Enter*** o el botón de la derecha para ver el resultado, que se mostrará en la parte inferior de la ventana del debugger.

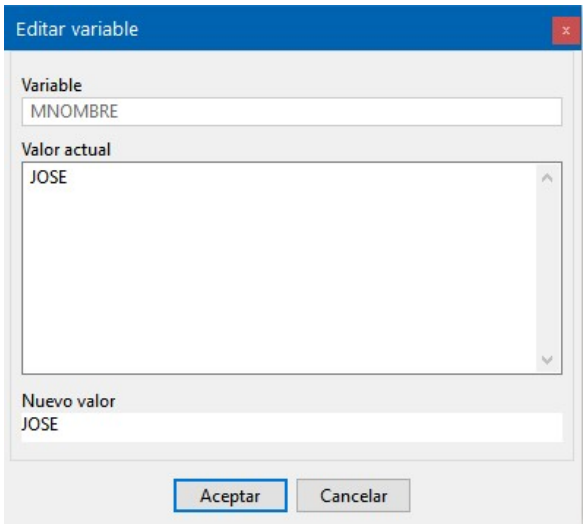


Si en vez de ***Enter*** pulsamos ***Ctrl+Enter*** o hacemos clic en el botón de más a la derecha, la expresión se añadirá al grupo *Watchpoints* y será evaluada cada vez que se detenga la ejecución del programa, incluido cuando estamos ejecutándolo paso a paso.



Cambiar el valor de una variable

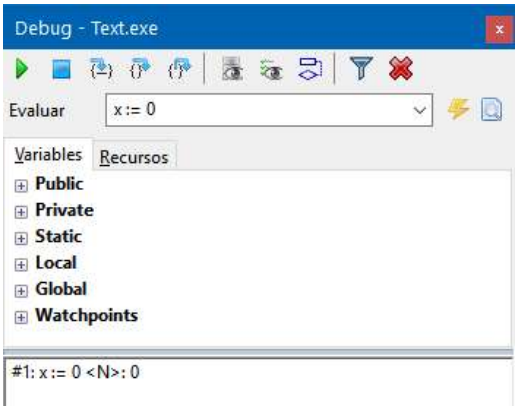
Cuando estemos inspeccionando una variable, podemos hacer doble clic sobre ella y se abrirá una ventana donde podemos cambiar su valor, siempre que dicho valor sea de tipo simple (carácter, número, fecha o lógico).



Si es de tipo complejo se abrirá otra ventana con los valores correspondientes, y desde ahí podremos volver a hacer doble clic y repetir la operación hasta que lleguemos a un valor de tipo simple.

El cambio de valor de una variable siempre será por otro valor del mismo tipo. Es decir, si estamos modificando una variable de tipo numérico, no podremos asignar un valor de tipo carácter o lógico, ni siquiera un *Nil*. Si quisiéramos cambiar el tipo lo podemos hacer utilizando el evaluador de expresiones.

P.ej., tenemos una variable *x* cuyo valor actual es *Nil*, y queremos cambiarlo por el número 0. En este caso bastará con escribir *x:=0* en el evaluador de expresiones.



Otras funciones del debugger

En la ventana del debugger tenemos también otros botones:

- *Inspeccionar áreas de trabajo*: nos permite ver toda la información relativa a las distintas áreas de trabajo DBF.
- *Inspeccionar valores SET*: muestra el estado de todos los indicadores de tipo SET del programa.
- *Ver pila de llamadas*: muestra toda la pila de llamadas (call stack) desde el inicio del programa hasta el punto donde esté detenido el depurador.
- *Filtrar los mensajes del depurador*: Nos permite mostrar solamente aquellas cadenas que hemos enviado desde nuestro programa, ocultando aquellas originadas en Windows, controladores de impresora, DLLs de terceros, etc.
- *Limpiar los mensajes del depurador*: Elimina todo los mensajes del área de mensajes del depurador.

Funciones auxiliares

Además de todo lo anterior, la librería XailerDebug incluye algunas funciones auxiliares para ayudarmos a depurar nuestro programa:

- ***Altd()***
  Nos permite invocar al debugger desde nuestro código. La ejecución del programa se detendrá justo en la línea siguiente.
- ***IsDebug() -> <lDebug>***
  Nos permite conocer si el programa está compilado en modo debug. Devolverá .T. si es así, aunque en ese momento no estemos depurándolo con XEdit.
- ***LogDebug(...)***
  Muestra el resultado de las expresiones que se pasen como parámetros en la ventana del debugger. Admite un numero variable de parámetros y de cualquier tipo.
- ***OutputDebugString(<cString>)***
  Es un "wrapper" de la función homónima del API de Windows. Envía la cadena pasada como parámetro a la ventana del debugger.

En bajo nivel, es decir, llamables desde C, también tenemos estas funciones:

- ***OutDebug(cString1 [, ...], NULL)***
  Es similar a LogDebug() pero en bajo nivel. Admite un número variable de parámetros, pero todos tienen que ser de tipo cadena. El último parámetro ha de ser necesariamente NULL.
- ***XA_OutputDebugString(cString)***
  Es equivalente a la función del API OutputDebugString()