Herramientas relacionadas con GiD

Depuración de código Tcl/Tk

La depuración de los errores de un programa típicamente ocupa un tiempo muy superior al de la propia escritura inicial del código. El uso de una herramienta apropiada para encontrar los defectos es por tanto fundamental, y uno de los puntos más débiles que ha tenido hasta ahora el lenguaje Tcl.

Por otra parte, una herramienta que durante la propia escritura del código ayuda a detectar fallos (comandos mal escritos, llaves que no cierran, etc) se convierte en un aliado insustituible para el desarrollador de programas.

Para avanzar en estos aspectos se ha desarrollado una herramienta lo más amigable posible: RamDebugger.

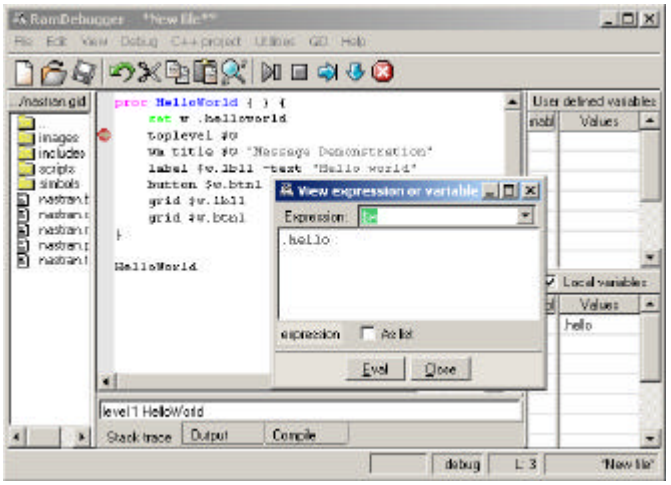


TABLA DE CONTENIDOS

1. DEPURACIÓN MANUAL CLÁSICA

Para realizar la depuración de errores (“debug”) de un programa en Tcl, pueden usarse varias técnicas:

La técnica más básica, pero en ocasiones necesaria, es imprimir el valor de las variables en un fichero.

En el fichero “scripts/ConfigureProgram.tcl” se puede asignar el valor “debug” a la variable global GidPriv(Configuration) (por defecto tiene el valor “release”). Entonces se pueden imprimir mensajes en un fichero “DebugGiDTcl.log” ubicado en el directorio “tmp” del sistema.

Para imprimir un mensaje puede usarse el procedimiento: DebugPrint $texto

Ó bien otro procedimiento algo más sofisticado, con algunos parámetros opcionales para medir tiempos o identar los mensajes según el nivel de la pila de procedimientos: DebugText -time -abs -rel -level $texto

También puede ser muy útil para depurar, imprimir un texto en una ventana, sin que se paralice el flujo, mediante: WarnWinText $texto, ó paralizando la ejecución (ventana modal), mediante: WarnWin $texto

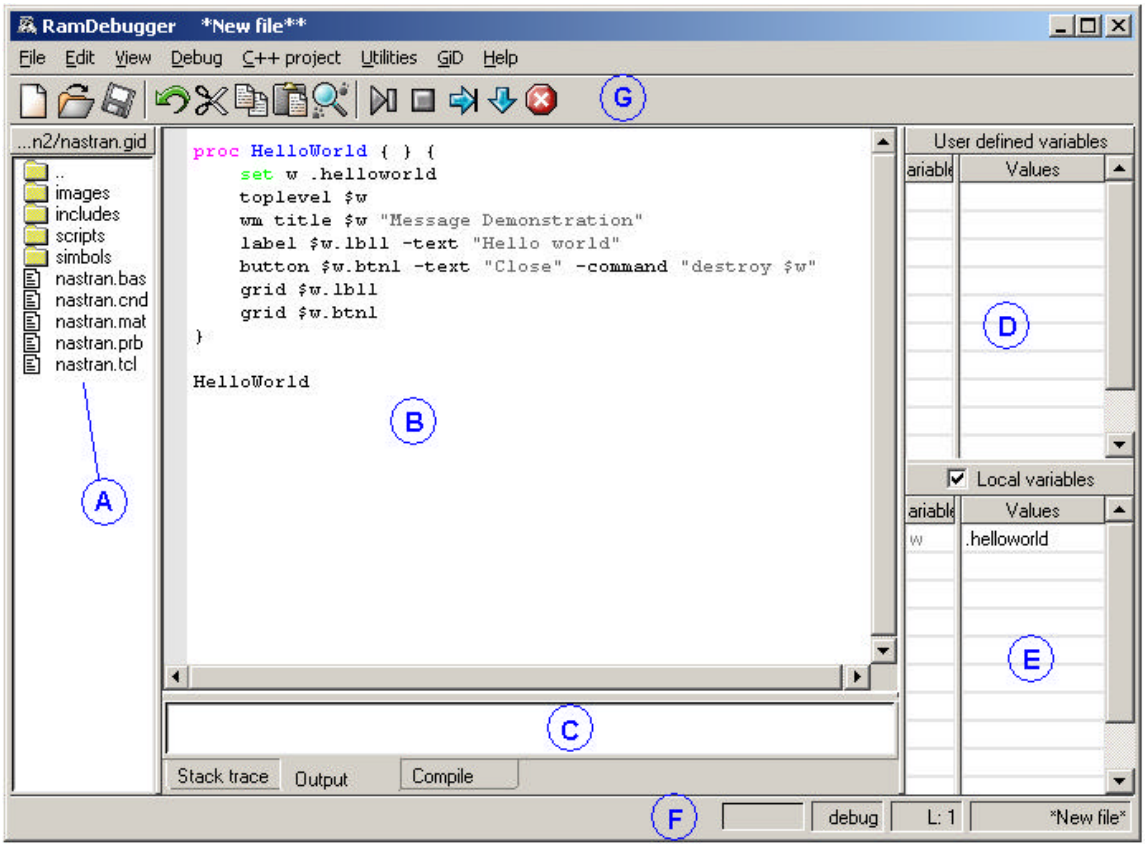
2. RAMDEBUGGER

Para una depuración más amigable, en Compass y CIMNE se ha desarrollado una herramienta especializada para editar y depurar código tcl: RamDebugger.

2.1. Descripción de la herramienta

RamDebugger está implementado completamente en Tcl/Tk, y en su actual distribución requiere que se instale una versión reciente del interprete Tcl (en futuras distribuciones probablemente se distribuirá como un ejecutable autosuficiente)

Para arrancar el programa, una vez instalado Tcl, basta hacer doble-clic sobre el fichero “RamDebugger.tcl”, con lo que se abre una ventana como la siguiente:



La distribución es similar a otras herramientas:

A - panel para la selección de ficheros

B - panel de visualización y edición del código

C -Mensajes del programa, errores,...

D – Visualización y modificación de las variables especificadas

E - Visualización y modificación de las variables del ámbito actual

F – Barra de mensajes (número de fila, etc)

G – Barra de iconos y menús de opciones.

El editor tiene una serie de prestaciones que lo hacen de por sí muy útil, incluso aunque no se desee depurar el código: Marcado por colores de los comandos, identación automática con el tabulador, mostrado de bloques al cerrar llaves, herramientas para deshacer, buscar y sustituir palabras, ayuda contextual de las palabras clave (tecla F1), comentar / descomentar regiones, Manual de referencia de Tcl/Tk y algunos “Packages” adicionales, etc.

2.2. Depuración local

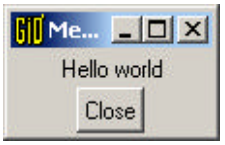
Para detener la ejecución en una línea, hay que situarse en ella y pulsar <F9>, con lo que aparece un punto de ruptura marcado con un círculo rojo junto a la línea.

Un “script” Tcl local se arranca desde el depurador pulsando la tecla <F5>.

Por ejemplo si se escribe el siguiente código, para crear una ventana Tk con una etiqueta con el clásico mensaje “Hello world” y un botón para cerrarla:

```
proc HelloWorld { } {
    set w .helloworld
    toplevel $w
    wm title $w "Message Demonstration"
    label $w.lbl1 -text "Hello world"
    button $w.btn1 -text "Close"  -command "destroy $w"
    grid $w.lbl1
    grid $w.btn1
}

HelloWorld
```



Se pone una parada en la 3ª línea (toplevel $w), y se ejecuta con <F5>, puede verse que la ejecución se para en dicha línea (marcada con un flecha amarilla), y en el panel E aparece la variable automática w con el valor actual “.helloworld”

Si quisiéramos podríamos cambiar su valor, por ejemplo a “.hellowin”

Si se mueve el cursor sobre el nombre de la variable en el panel de edición B, aparece una etiqueta mostrando el valor de la variable

Si se pulsa <F10> se continua la ejecución una línea, y si se pulsa <F11> se avanza una línea entrando en los procedimientos afectados (avance paso a paso).

Si se pulsa nuevamente <F5> la ejecución continua hasta la siguiente parada.

2.3. Depuración remota

En un caso, como por ejemplo en GiD, el código Tcl no es el que inicia la ejecución del programa, sino que el núcleo del programa arranca primero, e invoca posteriormente a los procedimientos tcl. Este hecho invalida una depuración local como la explicada anteriormente. Es necesario un mecanismo especial, mediante el cual, el programa se conecta a un programa en ejecución, y permite una depuración remota.

Por ejemplo, supongamos que arrancamos GiD, y tenemos un “tipo de problema” con un código similar al anterior, pero que es invocado por el procedimiento InitGIDProject, bien conocido por los desarrolladores de interfaces.

En este caso, tenemos un problemtype llamado “HelloWorld.gid”, que contiene únicamente el siguiente fichero “HelloWorld.tcl”

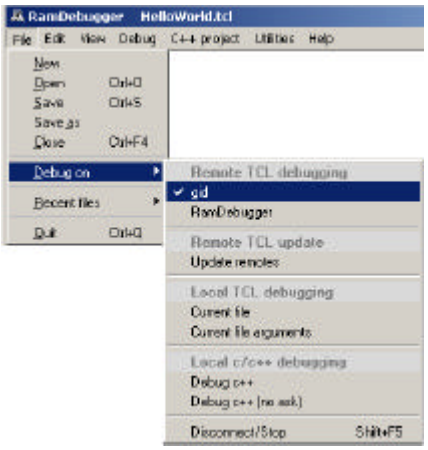
```
proc InitGIDProject { dir } {
    HelloWorld
}

proc EndGIDProject {} {
}

proc HelloWorld { } {
    set w .helloworld
    toplevel $w
    wm title $w "Message Demonstration"
    label $w.lbl1 -text "Hello world"
    button $w.btn1 -text "Close"  -command "destroy $w"
    grid $w.lbl1
    grid $w.btn1
}
```

Antes de ejecutar GiD, para hacer debugger remoto, hay que poner el valor de la variable GidPriv(Configuration) a “debug” (en el fichero “ConfigureProgram.tcl”).

Entonces ejecutamos el programa GiD, y después arrancamos RamDebugger, en el cual abrimos el fichero “HelloWorld.tcl”



En el menú de RamDebugger:

File->Debug On,

nos aparecerá la opción “gid”

Si no es así puede forzarse una actualización con el menú:

File->Debug On->Update remotes

Entonces podemos poner una parada con <F9>, por ejemplo en la única línea “HelloWold” del procedimiento InitGIDProject.

Ahora cargamos el tipo de problema en GiD (menú: Data->Problemtype->HelloWorld), y entonces el programa RamDebugger toma el control y detiene la carga en el punto especificado.

En ese momento, la depuración ya es igual que en el caso local, por ejemplo podemos entrar en el procedimiento HelloWorld con una ejecución paso a paso mediante la tecla <F11>.

En ocasiones puede ser útil usar la opción del menú: Debug->Reinstrument, para forzar una recarga del código tcl en GiD, por ejemplo después de realizar alguna modificación del código y una continuación de la ejecución sin reiniciar GiD.