LAUTERBACH
DEVELOPMENT TOOLS

Training Menu Programming

MANUAL

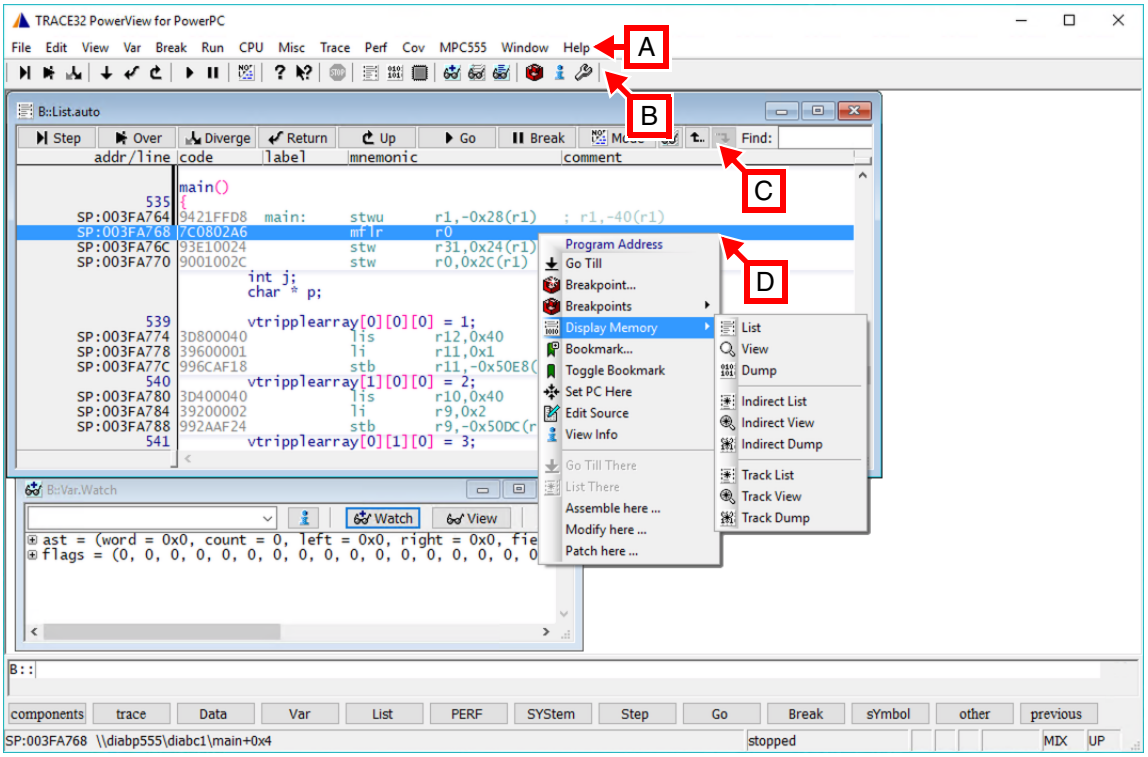
Training Menu Programming

Training Menu Programming

Version 06-Jun-2024

Customizable GUI Elements

Many elements of the TRACE32 PowerView user interface can be customized by the user. An overview is shown in the image below and the highlighted areas are those that may be customized.



A The main menu bar and accelerators

B The main toolbar

C Window local buttons

D Popup menus accessed via a right-click

The TRACE32 Default Menu

The default menu is located in a file called **t32.men**. When TRACE32 starts, it looks for this file in the current working directory and then in the system directory, which is pointed to by **SYS=** in the TRACE32 configuration file.

CPU selection will add further entries to the menu system. There is usually a CPU specific menu to access any on-chip peripherals but entries may also appear in other menus according to the chip's features.

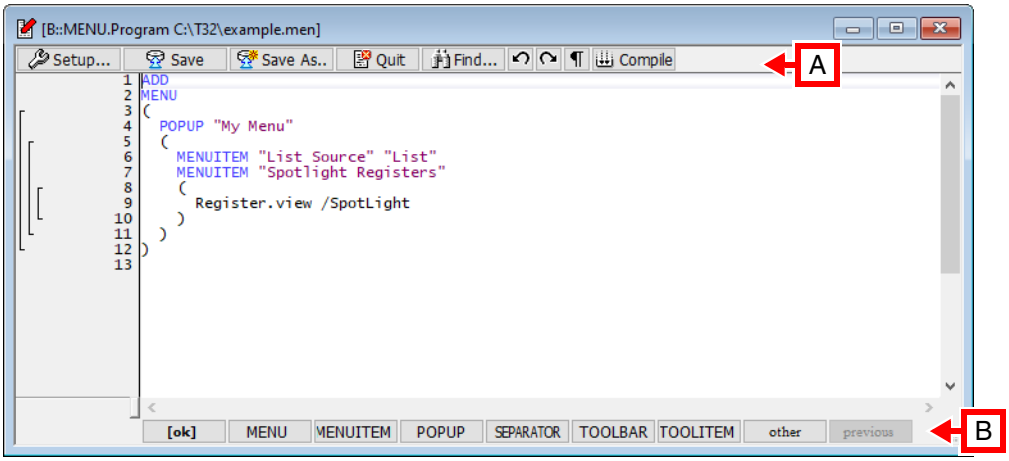
It is not recommended to update these files directly; a software update may change these files and then all local modifications will be lost.

We would recommend that you place your personal user interface customizations in the file **./work-settings.cmm**, where the "." represents the working directory from where TRACE32 was started. If you specified the user interface customizations for your team, then **~~/system-settings.cmm** is the correct place. The path prefix "**~~**" represents the TRACE32 system directory. Support for these settings files has been included since build 99518. For more details, please refer to **"Automatic Start-up Scripts"** in PRACTICE Script Language User's Guide, page 15 (practice_user.pdf).

If you are using older software, it is recommended to embed your user interface customizations within your PRACTICE start-up script (*.cmm).

The Built-in Menu Editor

Menus can be customized via a menu file. This is a text file with a **.men** extension. The command **MENU.Program** opens the built-in menu programming editor. It looks like this:



A Shortcut Buttons:
Save the menu file
Save the menu file with a different name
Save and close the menu file
Close the menu file without saving it
Save the menu file and compile it into TRACE32
Compile the menu file into TRACE32

B Softkeys to aid menu programming

The modifications can be immediately compiled into the TRACE32 user interface or the menu can be programmed at a later point either from the TRACE32 command line or in a PRACTICE script (*.cmm) by using:

```
MENU.ReProgram <menu_file>.men
```

Alternatively, the menu commands can be embedded into a PRACTICE script (*.cmm) file. An example is shown below.

```
MENU.RESet
MENU.ReProgram
(
   ADD
   MENU
   (
      POPUP "My Menu"
      (
         MENUITEM "List Source" "List"
         MENUITEM "Spotlight Registers"
         (
            Register.view /SpotLight
         )
      )
   )
)
ENDDO
```

The next section will explain the commands shown in more detail.

The menu system can be returned to the default values by using the command **MENU.RESet**.

Customizing the Main Menu Bar

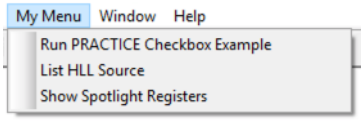
Adding a New Menu to the Main Menu Bar

First Example

The menu file below adds a new menu to the main menu bar.

```
ADD
MENU
(
   POPUP "My Menu"
   (
      MENUITEM "Run PRACTICE Checkbox Example"
      (
         DO "~~/demo/practice/dialogs/dialog_checkbox.cmm"
      )
      MENUITEM "List HLL Source" "List.Hll"
      MENUITEM "Show Spotlight Registers"
      (
         Register.view /SpotLight
         Step.Hll
      )
   )
)
```

When compiled into TRACE32, it looks like this:



Let's take a look at the concepts introduced in this script.

The first line of the script introduces the **ADD** command. This instructs TRACE32 that something is to be added to the menu system. Without this, the main menu bar and the toolbar will be replaced by what follows next.

The second line is **MENU**. This is used without parameters to tell TRACE32 that we wish to change the main menu bar. The changes to be made are defined in the following block. A block is defined by a pair of parentheses '(' and ')' and they must be on a line by themselves.

The drop-down menu itself is referred to as a **POPUP** and requires a name argument. This is the text that will appear at the top of the menu. If the name used already exists, then that **POPUP** will be adjusted. If the name does not exist, a new menu is created to the left of the **Window** menu. The contents of the menu are defined in the following block.

Finally, **MENUITEM** *<name> <command>* is introduced. The text passed as the *<name>* argument will appear on the **POPUP** as the menu entry and the *<command>* argument will be executed when the menu item is selected by the user. This command may be a single command enclosed in quotes or a block enclosed by parentheses. The example provided above shows how to target a single command, a block and how to call a PRACTICE script.

If a **MENUITEM** is marked as **DEFAULT** it is printed in bold. This will be the item selected if the user double-clicks the menu name. For example:

```
ADD
MENU
(
   POPUP "My Menu"
   (
      DEFAULT
      MENUITEM "Run PRACTICE Checkbox Example"
      (
         DO "~~/demo/practice/dialogs/dialog_checkbox.cmm"
      )
      MENUITEM "List HLL Source" "List.Hll"
      MENUITEM "Show Spotlight Registers"
      (
         Register.view /SpotLight
         Step.Hll
      )
   )
)
```

Creating New Shortcuts

Creative use of the *<name>* argument to a **MENUITEM** allows the possibility of creating new shortcuts for a TRACE32 command or command sequence. For example:

```
MENUITEM "List Hll Source,CTRL+L" "List.Hll"
```

Before we look at the details, we need some background information on the TRACE32 defined shortcuts.

The shortcut **ALT+***<letter>* is reserved for the TRACE32 default menu. For example:

ALT+b	Opens the **Break** menu.
ALT+b+s	Opens the **Break.Set** dialog window.

If you hold the **ALT** key, you can see which letters are reserved for the main menus.

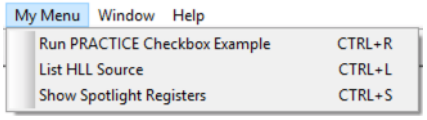
The reserved letters within each drop-down menu are underlined as well.

The function keys are also reserved for the TRACE32 default menu. **F1** is used to get help and support and **F2** to **F9** are used by the **Run** menu.

Allowing for those restrictions, it is possible to create user-defined shortcut key sequences when creating a **MENUITEM**. To do this, add a comma and then the shortcut key sequence as part of the **MENUITEM**'s *<name>* argument. Some examples are shown in the code snippet below.

```
ADD
MENU
(
   POPUP "My Menu"
   (
      MENUITEM "Run PRACTICE Checkbox Example,CTRL+R"
      (
         DO "~~/demo/practice/dialogs/dialog_checkbox.cmm"
      )
      MENUITEM "List HLL Source,CTRL+L" "List.Hll"
      MENUITEM "Show Spotlight Registers,CTRL+S"
      (
         Register.view /SpotLight
         Step.Hll
      )
   )
)
```

The results will look like this:

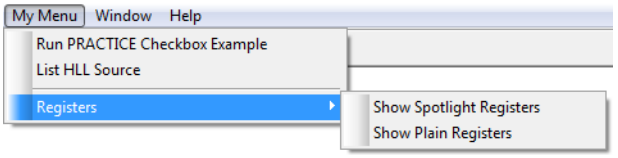


Including Sub-Menus

Adding a sub-menu to a drop-down menu is done by adding another **POPUP** within a **POPUP** block. The sub-menu has been highlighted in the example below.

```
ADD
MENU
(
   POPUP "My Menu"
   (
      MENUITEM "Run PRACTICE Checkbox Example"
      (
         DO "~~/demo/practice/dialogs/dialog_checkbox.cmm"
      )
      MENUITEM "List HLL Source" "List.Hll"
      SEPARATOR
      POPUP "Registers"
      (
         MENUITEM "Show Spotlight Registers"
         (
            Register.view /SpotLight
         )
         MENUITEM "Show Plain Registers" "Register.view"
      )
   )
)
```

This example also introduces the **SEPARATOR** command. This places a separator on the menu in the required position. This menu looks like the image below.



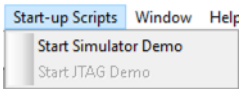
Conditional Menu Items

Sometimes, a menu item needs to be enabled or disabled based upon a runtime check. The command **ENABLE** *<condition>* will do this for the next menu item.

If the condition evaluates to TRUE, the following menu item will be enabled.

```
ADD
MENU
(
   POPUP "Start-up Scripts"
   (
      ENABLE (INTERFACE.SIM()==TRUE())
      MENUITEM "Start Simulator Demo" "DO demo_sim.cmm"
      ENABLE (hardware.ICD()==TRUE())
      MENUITEM "Start JTAG Demo" "DO demo_jtag.cmm"
   )
)
```

If the TRACE32 Instruction Simulator is detected, the **MENUITEM "Start JTAG Demo"** will be disabled.



The function **INTERFACE.SIM()** returns true if TRACE32 detects that it is configured as an instruction set simulator.

The function **hardware.ICD()** returns true of TRACE32 detects one of the universal base modules.

Altering an Existing Drop-down Menu

By specifying the name of an existing drop-down menu with the **POPUP** command, it is possible to change items of the TRACE32 default menus.

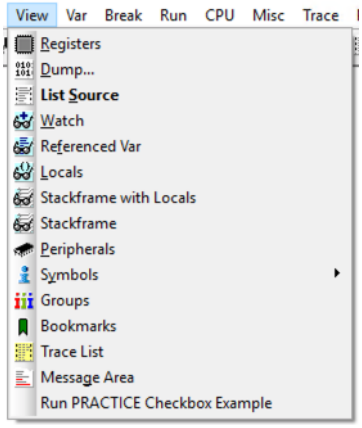
Add a New Item to an Existing Drop-down Menu

The example shown below will add a new menu to the existing **View** drop-down menu.

```
ADD
MENU
(
   POPUP "&View"
   (
      MENUITEM "Run PRACTICE Checkbox Example"
      (
         DO "~~/demo/practice/dialogs/dialog_checkbox.cmm"
      )
   )
)
```

The **MENUITEM "&View"** has to start with an ampersand (&) because **ALT+v** is the shortcut for the existing **View** menu. Care should be taken to ensure that shortcuts match when editing existing drop-down menus.

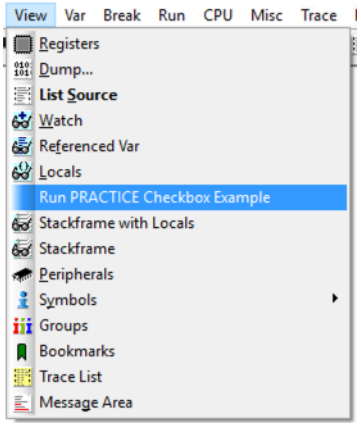
The new menu item will appear at the bottom of the existing menu, as seen in the image below.



To change the position of the new item, use the keywords **AFTER** and **BEFORE**. Items placed on a menu following an **AFTER** command will appear below the specified item on the drop-down, for example:

```
ADD
MENU
(
    POPUP "&View"
    (
        AFTER "&Locals"
        MENUITEM "Run PRACTICE Checkbox Demo"
        (
            DO "~~/demo/practice/dialogs/dialog_checkbox.cmm"
        )
    )
)
```

This will place the new menu item immediately below the **Locals** menu item, causing all other items to be shifted down the menu by one place. The results can be seen below.

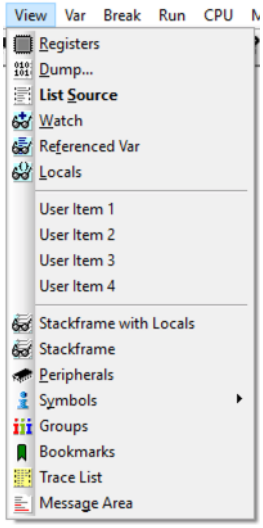


Menu items placed following a **BEFORE** command will be located immediately above the specified item on the drop-down menu.

The keywords **BEFORE** and **AFTER** can be used in any order to create more complex menus, for example:

```
ADD
MENU
(
   POPUP "&View"
   (
      AFTER "&Locals"
      MENUITEM "User Item 1" ""
      AFTER "User Item 1"
      MENUITEM "User Item 2" ""
      AFTER "User Item 2"
      MENUITEM "User Item 3" ""
      AFTER "User Item 3"
      MENUITEM "User Item 4" ""

      BEFORE "User Item 1"
      SEPARATOR
      AFTER "User Item 4"
      SEPARATOR
   )
)
```

Which looks like this:



Change an Existing Menu Item

By specifying an existing item of an existing drop-down menu and using the **REPLACE** command, it is possible to change the command that is executed when that menu item is selected by the user. For example, to change the **View** menu -> **Locals** menu item to always enable to **SpotLight** option:

```
ADD
MENU
(
   POPUP "&View"
   (
      REPLACE
      MENUITEM "[:varlocal]&Locals"
      (
         Var.Local %SpotLight
      )
   )
)
```

The text **[:varlocal]** makes sure that the icon for the menu item is retained. Refer to **"The Built-in Icon Library"**, page 19 for more details about the icons pre-defined by TRACE32.

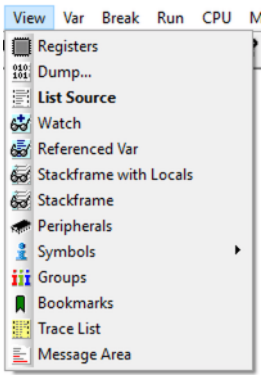
"&Locals" must start with an ampersand to preserve the shortcut key sequence.

Delete an Item from an Existing Drop-down Menu

By using the **DELETE** command, it is possible to delete an item from an existing drop-down menu. For example: to remove the **Locals** item from the **View** menu.

```
ADD
MENU
(
   POPUP "&View"
   (
      DELETE "&Locals"
   )
)
```

The results look like this:

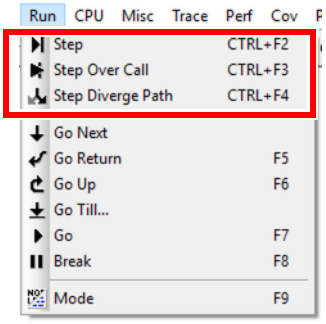


Change Shortcuts of the Run Menu

Similarly, the shortcut key sequence can be altered to suit an individual's preferences. For example:

```
ADD
MENU
(
   POPUP "&Run"
   (
      REPLACE
      MENUITEM "[:step]&Step,CTRL+F2" "Step.single"
      REPLACE
      MENUITEM "[:stepover]Step &Over Call,CTRL+F3" "Step.Over"
      REPLACE
      MENUITEM "[:stepdiverge]Step &Diverge Path,CTRL+F4" "Step.Diverge"
   )
)
```

The before and after images can be seen below, with the changes highlighted.



The entries **`[:step]`**, **`[:stepover]`** and **`[:stepdiverge]`** are there to ensure that the icons for the menu items are retained. For more information, please refer to **"The Built-in Icon Library"**, page 19.

Customize the Toolbar

The toolbar is the area with icons in it that sits below the main menu bar in the TRACE32 window. This can also be modified.

Add a New Button to the Toolbar

When adding a new item to the toolbar, use the **ADD** command with the **TOOLBAR** option. Each new item is a **TOOLITEM**.

TOOLITEM *<balloon_text> <icon> <command>*

<balloon_text>	The text that is displayed when the user hovers the mouse pointer over the icon.
<icon>	Two letters, a comma and a third letter. The first two letter are converted into a bitmap. The letter after the comma indicates the color which they will be displayed in. The color letters are the same as those used with the **BITMAPEDIT** command.
<command>	The command that is executed as a result of the button being clicked. It may be a single command enclosed in quotes or a block.

The concepts can be seen in the example below.

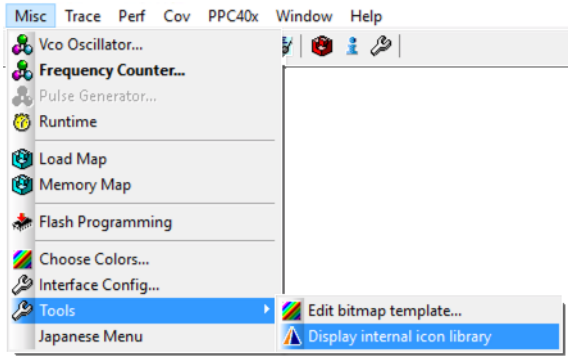
```
ADD
TOOLBAR
(
   TOOLITEM "Set PC to main()" "PC,R" "Register.Set PC main"
   SEPARATOR
   TOOLITEM "Open Windows" "WN,B"
   (
      WinPOS 1.25 0.14 92. 20. 14. 1. W001
      WinTABS 10. 10. 25.
      List.auto

      WinPOS 100. 38. 49. 18. 0. 0. W002
      Register.view

      WinPOS 2.5 26.4 76. 12. 0. 0. W003
      Var.Local
   )
)
```

When compiled, this will add two new buttons and a separator to the toolbar and will look like the image below.

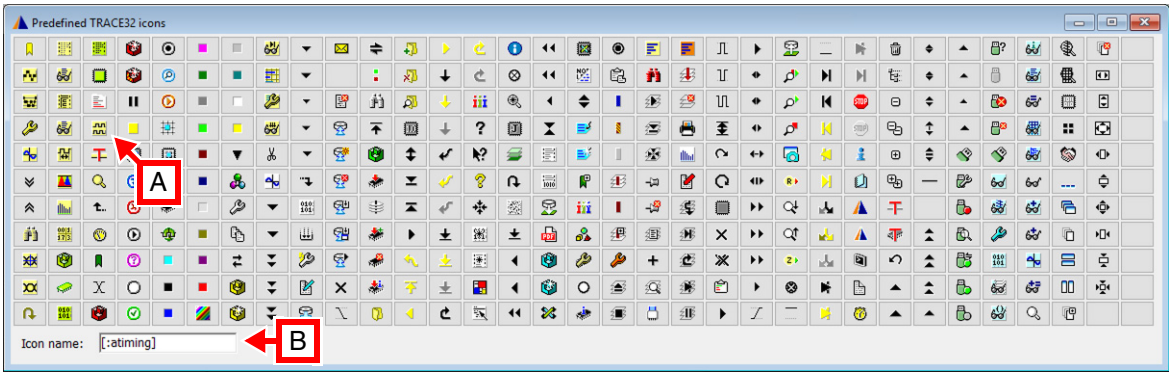
The Built-in Icon Library

TRACE32 includes a library of built-in icons which can be used in all menus, toolbars and dialogs. Each icon is represented by a code. The icons and their corresponding codes can be found by opening:

- **Misc** menu -> **Tools** -> **Display Internal icon Library**.



The results look like this.

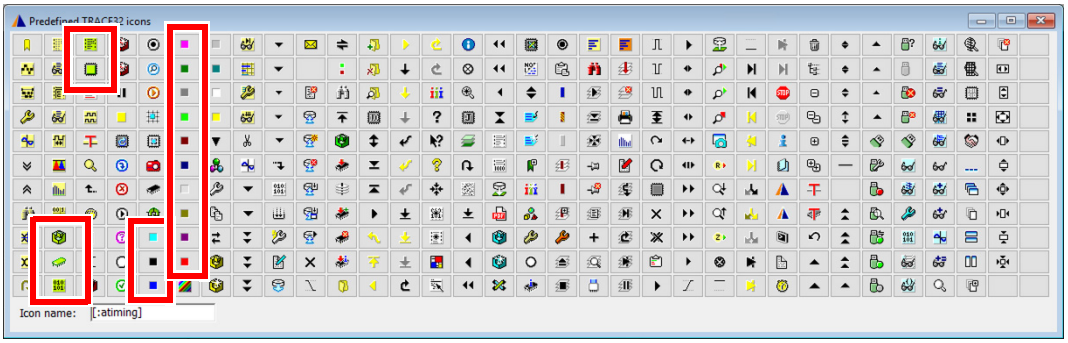


Clicking an icon [**A**] causes the code for that icon to be displayed [**B**].

In order to avoid confusion we recommend that the icons retain their functional meaning. For example, use the **`[:practice]`** icon when adding a menu item or tool item that runs a script.

```
ADD
MENU
(
   POPUP "My Tools"
   (
      MENUITEM "[:practice]Run PRACTICE Checkbox Example"
      (
         DO "~~/demo/practice/dialogs/dialog_checkbox.cmm"
      )
   )
)
```

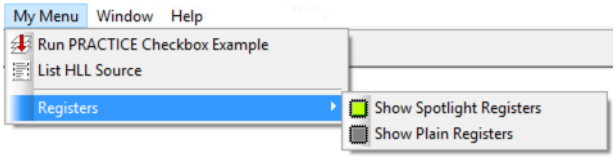
For new or extended functions, we recommend that you use one of the rarely used icons, for example the **[:apu*]** or **[:color*]** icons. These are highlighted in the image below.



To use one of the icons in a menu, place the code for it into the string that will appear as the menu entry. An example is shown below with the codes highlighted.

```
ADD
MENU
(
    POPUP "My Menu"
    (
        MENUITEM "[:practice]Run PRACTICE Checkbox Example"
        (
            DO "~~/demo/practice/dialogs/dialog_checkbox.cmm"
        )
        MENUITEM "[:list]List HLL Source" "List.Hll"
        SEPARATOR
        POPUP "Registers"
        (
            MENUITEM "[:apureg]Show Spotlight Registers"
            (
                Register.view /SpotLight
            )
            MENUITEM "[:reg]Show Plain Registers" "Register.view"
        )
    )
)
```

Compiling this will produce:



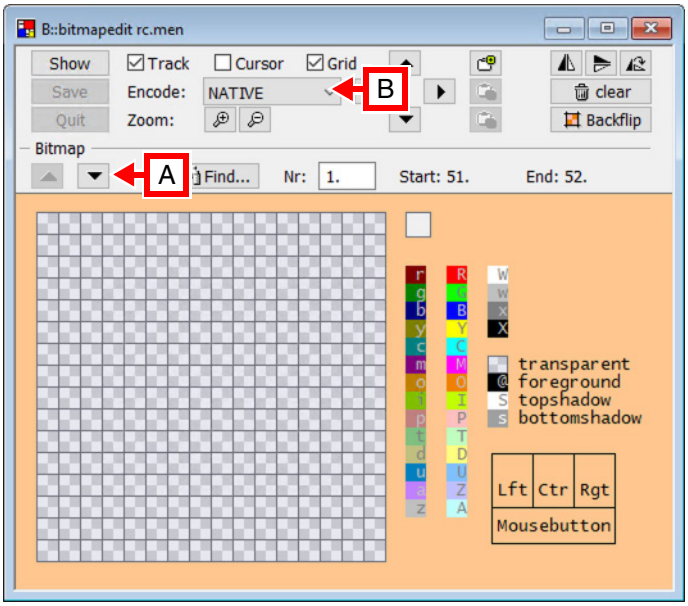
The procedure is very similar for using built-in icons with toolbar items. Place the code for the icon into the argument where the bitmap image for the toolbar item would be. For example:

```
ADD
TOOLBAR
(
   TOOLITEM "Set PC to main()" "[:btnapply]" "Register.Set PC main"
)
```

Compiling this menu will add a new button to the toolbar as shown below.

Creating Custom Icons

The command **BITMAPEDIT** can be used to create custom icons for TRACE32 menus and toolbars. It requires a menu or script file as an argument and when run it looks like this.



It provides a convenient way for users to draw an image which can be used as a tool item or menu item icon.

BITMAPEDIT requires placeholders for each icon within the script or menu file. These are a matched pair of square brackets '[' and]' with a single space between them. Use the up and down arrows (labeled [**A**] in the image above) to navigate between bitmaps or placeholders.

Once a bitmap has been drawn, save the file by clicking the **Save** button. It is recommended to save with ENCODE set to **SIGNATURE** instead of **NATIVE**. This is highlighted by point [**B**] in the image above.

To Add a Custom Icon to a Menu

To add a custom icon to a menu:

1. Ensure the placeholders are in the correct place in the menu file. For example:

```
ADD
MENU
(
   POPUP "My Menu"
   (
      MENUITEM "[ ]Run PRACTICE Checkbox Example"
      (
         DO "~~/demo/practice/dialogs/dialog_checkbox.cmm"
      )
      MENUITEM "[ ]List HLL Source" "List.Hll"
      SEPARATOR
      POPUP "Registers"
      (
         MENUITEM "[ ]Show Spotlight Registers"
         (
            Register.view /SpotLight
         )
         MENUITEM "[ ]Show Plain Registers" "Register.view"
      )
   )
)
```

2. Load the menu file with **BITMAPEDIT** *<filename>*
3. Create the icons.
4. Save the file.
5. Reprogram the menu or run the script.

To add a custom icon to a toolbar:

1. Replace the three letter sequence that creates an icon with a pair of empty square brackets. An example is shown here:

```
ADD
TOOLBAR
(
   TOOLITEM "Set PC to main()" "[ ]" "Register.set PC main"
   SEPARATOR
   TOOLITEM "Open My Windows" "[ ]"
   (
      WinPOS 1.25 0.14 92. 20. 14. 1. W001
      WinTABS 10. 10. 25.
      List.auto

      WinPOS 100. 38. 0.28 49. 18. 0. 0. W002
      Register.view

      WinPOS 2.5 26.4 76. 12. 0. 0. W003
      Var.Local
   )
)
```

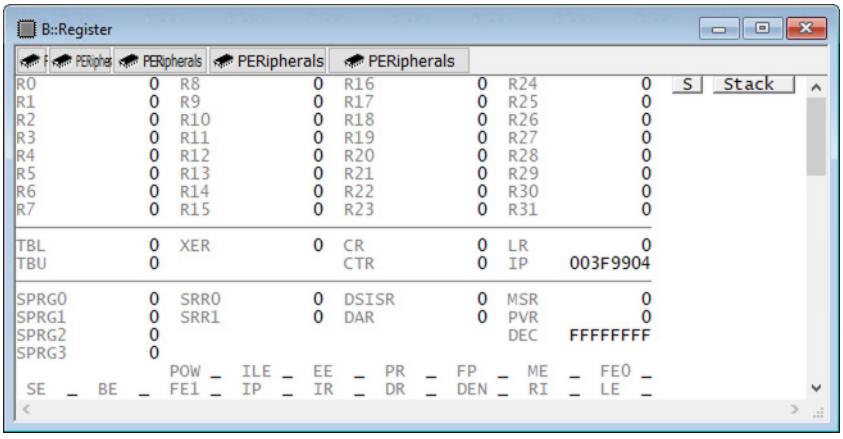
2. Load the file with **BITMAPEDIT**.
3. Create the icons.
4. Save the file.
5. Reprogram the menu or run the script.

Adding Local Buttons to a Window

Most windows within TRACE32 can have a series of toolbar-like buttons added to them. Use the command **BUTTONS** with **ADD**. For example:

```
ADD
BUTTONS "R."
(
   WIDTH SMALLEST
   MENUITEM "[:chip]Peripherals" "PERipherals.view"
   WIDTH SMALLER
   MENUITEM "[:chip]Peripherals" "PERipherals.view"
   WIDTH NORMAL
   MENUITEM "[:chip]Peripherals" "PERipherals.view"
   WIDTH WIDER
   MENUITEM "[:chip]Peripherals" "PERipherals.view"
   WIDTH WIDEST
   MENUITEM "[:chip]Peripherals" "PERipherals.view"
)
```

The results will look like this:



The **BUTTONS** command requires the shortcut string for the window to which it will be applied. Here we used "R.", which is the shortcut for "Register.".

Each button needs a **WIDTH** value before it can be placed. The values range in ascending order of size: SMALLEST, SMALLER, NORMAL, WIDER, WIDEST.