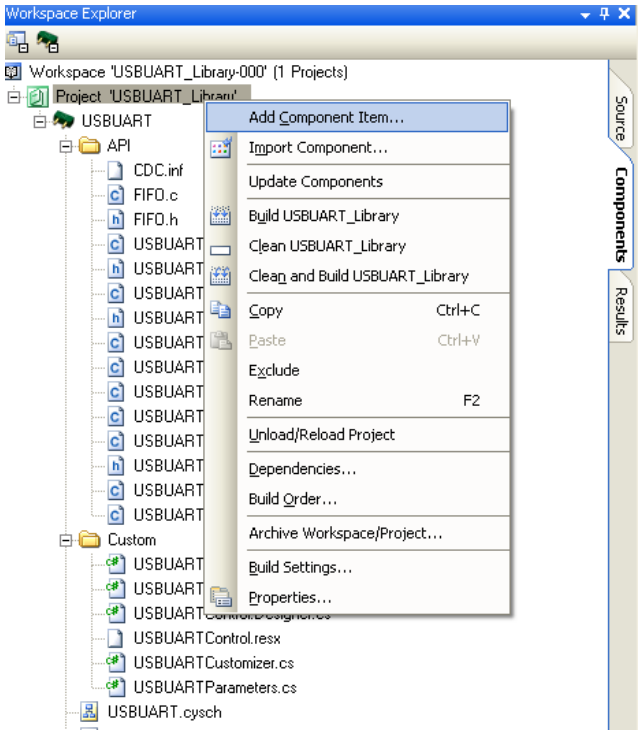
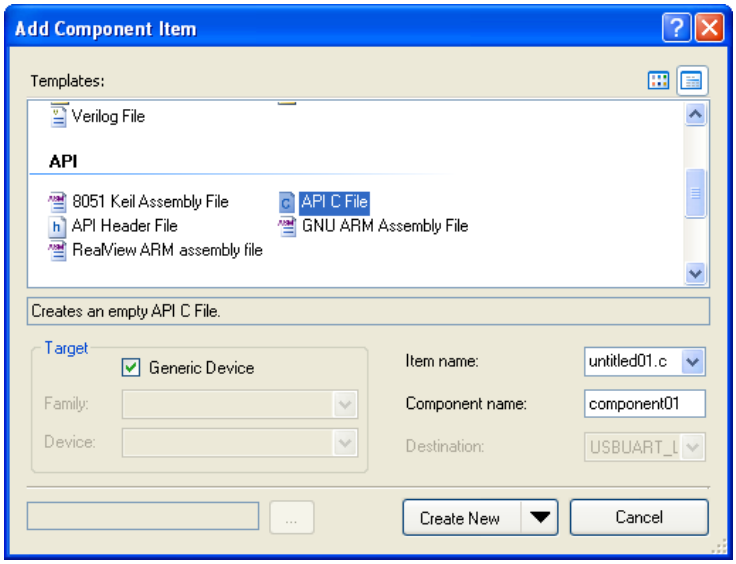
Q. What is the procedure to add APIs to a component?

A. Following is the process for adding APIs to a component.

1. Right click on the component name in the Workspace explorer and select Add Component Item.



The Add Component Item dialog displays.



2. Select the icon for the component item you wish to add.
3. Select the Target option.

Note: You can specify whether the component item is for a generic device or a specific family and/or device.

4. Type in the Item name.
5. Click Create New.

The component item displays in the Workspace Explorer. Depending on the target options you specified, the component item may be located in a subdirectory.

6. Repeat the process to add more API files.

Q. Is it necessary that a component should always have APIs to support its functionality?

A. Though it is not necessary, it is highly recommended to create APIs for the component. It has to be written with respect to Instance name. Following are the recommended APIs for any component:

1. `$INSTANCE_NAME`_Start()
2. `$INSTANCE_NAME`_Stop()
3. `$INSTANCE_NAME`_Sleep()
4. `$INSTANCE_NAME`_Wakeup()
5. `$INSTANCE_NAME`_SaveConfig ()
6. `$INSTANCE_NAME`_RestoreConfig()
7. `$INSTANCE_NAME`_Init()
8. `$INSTANCE_NAME`_Enable()
9. `$INSTANCE_NAME`_SetPower() (If any Analog Block is being used in the component)

Q. How different source files and components placed in the schematic are linked together?

A. The source files control the behavior of the components by calling APIs that regulate their functioning. These APIs can enable or disable the components and cause differences in device behavior. Here is a code snippet from a project source file which invokes a number of component APIs.

```
  /* Initialization Code: */
CYGlobalIntEnable;
  USBUART_1_Start(0, USBUART_1_5V_OPERATION);
  while(!USBUART_1_bGetConfiguration());
  USBUART_1_Init1();

  /* Main Loop: */
for(;;)
```

```
    {
            Count = USBUART_1_bGetRxCount();
            if(Count != 0)                                          /*
Check for input data from PC */
            {
                  USBUART_1_ReadAll(Buffer);
                  USBUART_1_Write(Buffer, Count);           /* Echo
data back to PC */
                  while(!USBUART_1_bTxIsReady()){}          /* Wait for
Tx to finish */
            }
    }
```

Q. How to write Low Power APIs for any Custom component?

A. The low power APIs for a component are meant to keep the component configuration intact while the PSoC 3/5 device goes to sleep mode. Also, sleep APIs should enable the user to keep the component in sleep mode Following are the low power APIs for a component:

1. **`$INSTANCE_NAME`_SaveConfig():** Save the component configuration to retain the state across the low power state of PSoC 3/5 device. In particular, a backup should be taken for those registers which do not retain the state across low power modes.
2. **`$INSTANCE_NAME`_RestoreConfig():** Restore the component configurations that was saved by the _SaveConfig API.
3. **`$INSTANCE_NAME`_Sleep():** Put the component in sleep and save the configuration
4. **`$INSTANCE_NAME`_WakeUp():** Restore the saved configuration and enable the component.

You can refer to the Section 11.4 – Low Power Support of Component Author Guide which has a detailed explanation of how these APIs should be implemented.

Q. How to access the APIs of the components which are used in designing the custom component?

A. To access the APIs of components which are used in designing the custom components; following pattern should be observed:

`$INSTANCE_NAME`_<COMPONENT_NAME>_<API>();

where; <COMPONENT_NAME> is the name of component used in custom component
<API> is the name of API.

For example, if a timer is used with instance name ‘Timer_1’ to design a customer component, following call will start the timer:

```
`$INSTANCE_NAME`_Timer_1_Start();
```