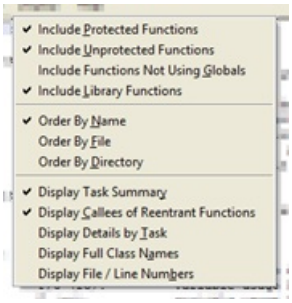
Reentrant Functions

Functions called from within more than one task are listed in the Reentrant Functions report. To the extent that these functions set or read persistent variables, such as globals, statics, or local-statics, then calling these functions can result in unanticipated interactions with other tasks that also call these functions. Consider the following example:




```
int globalZ1, globalZ2;

void funcC() {}

void funcB() {
    int localB;
    globalZ2 = 2;
    /* some calculations */
    localB = globalZ2;
}

void funcA2() {}

void funcA1() {
    int localA1;
    globalZ1 = 1;
    /* some calculations */
    localA1 = globalZ1;
}

void funcA() {
    funcA1();
    funcA2();
}

void taskX() {
    funcA();
    DisableInt();
    funcB();
    EnableInt();
    funcC();
}

void taskY() {
    funcA();
    DisableInt();
    funcB();
    EnableInt();
    funcC();
}
```

The Reentrant Functions report uses the terms member, reentrant, and callee functions. Members are the full set of functions making up a task, consisting of the task root function, and all the functions called directly or transitively from that root. Shared functions are member functions common between two or more tasks, and are categorized as either reentrant or callee functions. Reentrant functions are those shared functions that are called by one or more non-shared member functions. These might be thought of as the root shared functions. Callee functions are shared functions which are only called by other shared functions, and thus do not represent an entry into the shared function calling hierarchy.

In listing each reentrant function, the report indicates not only the tasks that call each reentrant function, but also whether or not that function call was made from inside a critical region. This analysis supports a review of potential interactions between tasks.

```
Reentrant Functions

Settings:
        Critical Region:              CR1 ( DisableInt / EnableInt )

        Protected Functions:          displayed
        Unprotected Functions:        displayed
        Functions not Using Globals:  displayed
```

```
        Library Functions:            displayed

Task Definitions
Tasks are from User Defined Tasks
Name            Members    Reentr   Callees  Root
TaskX             8 [+]     3 [+]     2 [+]  taskX
TaskY             8 [+]     3 [+]     2 [+]  taskY

Reentrant Function                                File (Line)
    Callees
          Task
             Line Number of Usage
                  Critical Region
                               User of Reentrant Function

funcA                                             reentr_funcs.c (21)
    funcA1                                            reentr_funcs.c (14)
    funcA2                                            reentr_funcs.c (12)
          TaskX
               27 U   (CR1) taskX                                      reentr_funcs.c (26)
          TaskY
               35 U   (CR1) taskY                                      reentr_funcs.c (34)

funcB                                             reentr_funcs.c (5)
          TaskX
               29 P   (CR1) taskX                                      reentr_funcs.c (26)
          TaskY
               37 P   (CR1) taskY                                      reentr_funcs.c (34)

funcC                                             reentr_funcs.c (3)
          TaskX
               31 U   (CR1) taskX                                      reentr_funcs.c (26)
          TaskY
               39 U   (CR1) taskY                                      reentr_funcs.c (34)
```

You're able to filter the results, focusing on those reentrant functions most likely to cause problems. Functions, such as funcC, that do not use any persistent, global variables, are safe and by default are omitted from the reported results.

Global variable use that does occur can be protected through the use of critical regions. For example, the interrupt protection surrounding the calls to funcB protects against globalZ2 from being modified or read by a different task between the time when globalZ2 is set and the time it is used. The use of reentrant functions within protected critical regions is generally safer than that in unprotected regions, and can also be filtered out. Here is the same report with these two filters applied.

```
Reentrant Functions

Settings:
        Critical Region:              CR1 ( DisableInt / EnableInt )

        Protected Functions:          omitted
        Unprotected Functions:        displayed
        Functions not Using Globals:  omitted
        Library Functions:            displayed

Task Definitions
Tasks are from User Defined Tasks
Name            Members    Reentr   Callees  Root
TaskX             8 [+]     1 [+]     2 [+]  taskX
TaskY             8 [+]     1 [+]     2 [+]  taskY

Reentrant Function                                File (Line)
    Callees
          Task
             Line Number of Usage
                  Critical Region
                               User of Reentrant Function

funcA                                             reentr_funcs.c (21)
    funcA1                                            reentr_funcs.c (14)
```

```
TaskX
     27 U   (CR1) taskX                                    reentr_funcs.c (26)
TaskY
     35 U   (CR1) taskY                                    reentr_funcs.c (34)
```