Rhapsody®

Using Rhapsody's Interrupt-Driven Framework (IDF) White Paper

Before using the information in this manual, be sure to read the “Notices” section of the Help or the PDF available from **Help > List of Books**.

This edition applies to Telelogic Rhapsody 7.4 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

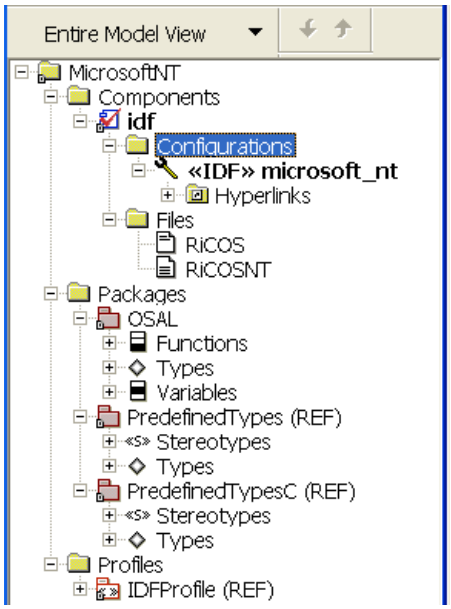
Rhapsody's Interrupt-Driven Framework (IDF)

For systems requiring a solution with a smaller footprint, the OXF provided with Rhapsody in C is not appropriate. To provide a solution for these environments, a limited framework called IDF (Interrupt-Driven Framework) is also provided with Rhapsody. Refer to the **Limitations of the IDF** section for Rhapsody's IDF restrictions.

Creating a Sample IDF Project

To learn about the Rhapsody IDF, several IDF components are included as samples that can be adapted for different target systems. To use the IDF features in sample projects, follow these steps:

1. Change the **Properties** of the *S*`hare\LangC\idf\Adapters\Microsoft` to remove the "Read-only" restriction and apply this change to the subdirectories. Clicking **OK** to save these changes.
2. Start Rhapsody in C.
3. Open the *S*`hare\LangC\idf\Adapters\Microsoft\MicrosoftNT.rpy` project.



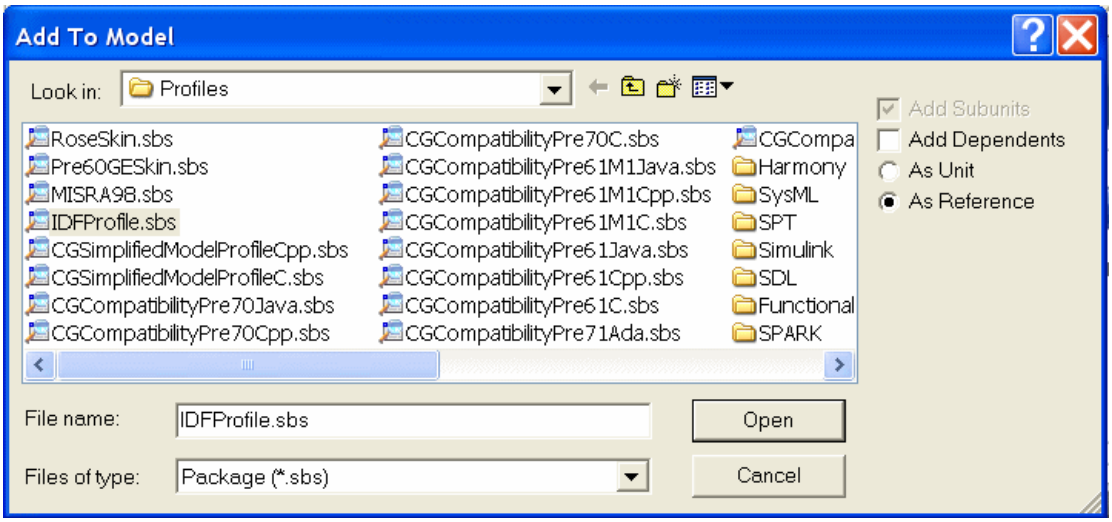
4. Check to be certain the correct "idf" and "microsoft_nt" components are selected (as shown below).

5. Click  to generate and make the code. This builds the `MSidf.lib` library in the `Share\LangC\lib` directory. Save and close the `MicrosoftNT.rpy` project. Rhapsody remains open.

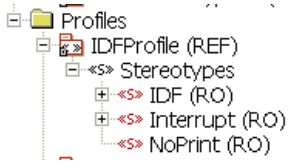
6. In Rhapsody's `Samples\CSamples` directory, open the `CycleComputer.rpy` project.

7. Select the **File > Add to Model** option to display the Add To Model dialog box.

8. Select `Package (*.sbs)` in the **Files of Type** field. In the folder selection area, navigate to the **Rhapsody <version>\Share\Profiles** folder and highlight the `IDFProfile.sbs` file and select **As Reference**, as shown in this example. Click **Open** to add this .sbs file to the CycleComputer model.

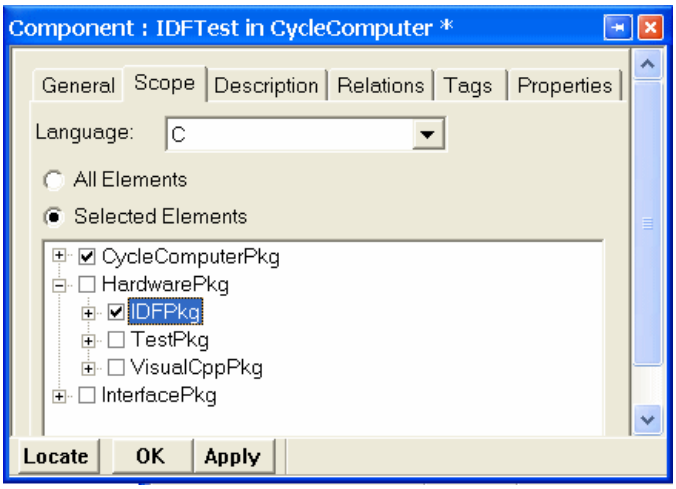


Note that IDFProfile (REF) is now in the Profiles folder in the browser, as shown below.



9. In the browser, highlight the **Components** item, right click, and select **Add New Component**. Type `IDFTest` into the area provided for the new component in the browser.

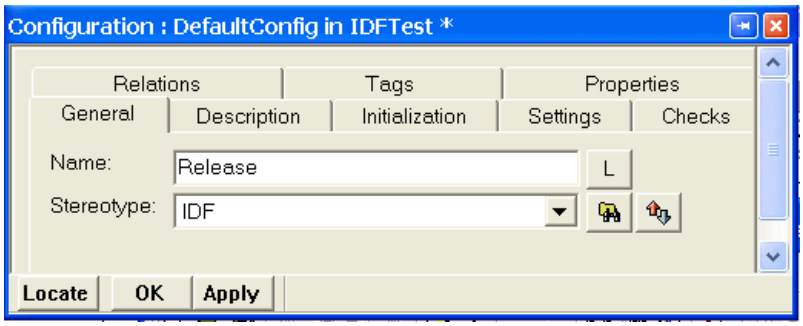
10. Right-click the new `IDFTest` component and select **Features**. In the **Scope** tab of the features dialog box, select the `CycleComputerPkg` and the `IDFPkg` under the HardwarePkg (as shown below) and click **OK**.



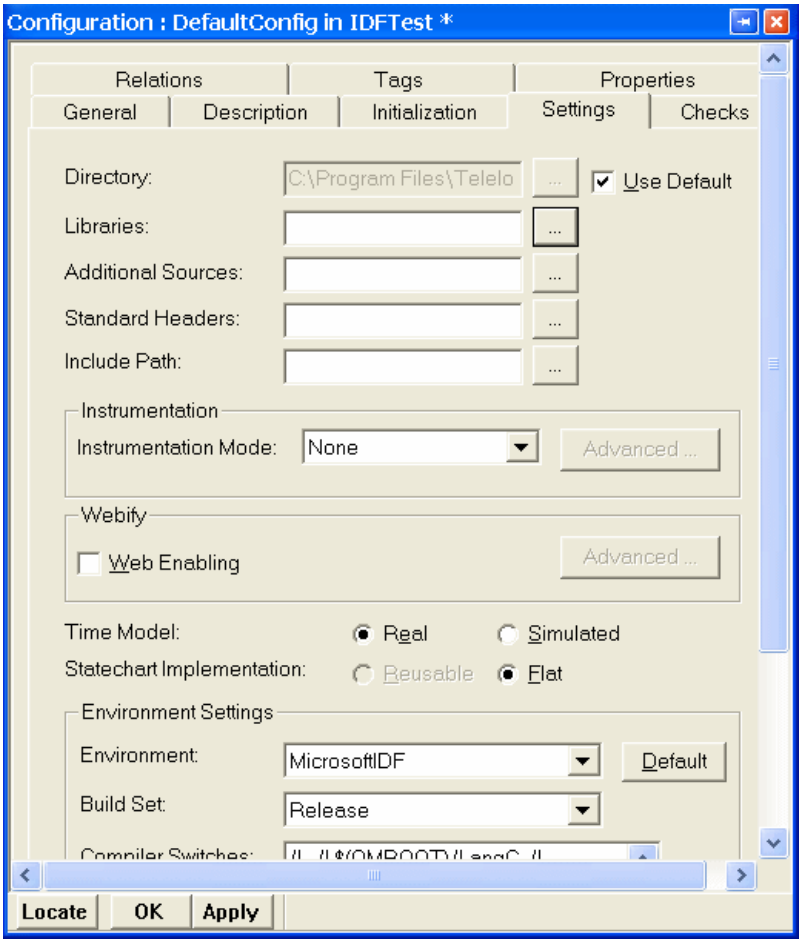
11. In the browser under the new `IDFTest` component, right click the `DefaultConfig` to display the Features dialog box.

12. In the General tab, change the **Name** of the configuration to be `Release`. Beside the **Stereotype** field, click the  and select **IDF** from the Profiles tree.
    You must apply the IDF stereotype to the configuration to ensure that the properties necessary for using the IDF are set correctly.

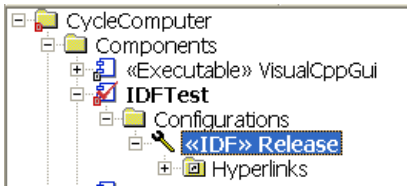
13. Click **OK** to return to the General tab. Click **Apply** to save the new configuration name and IDF stereotype and keep the dialog box open.



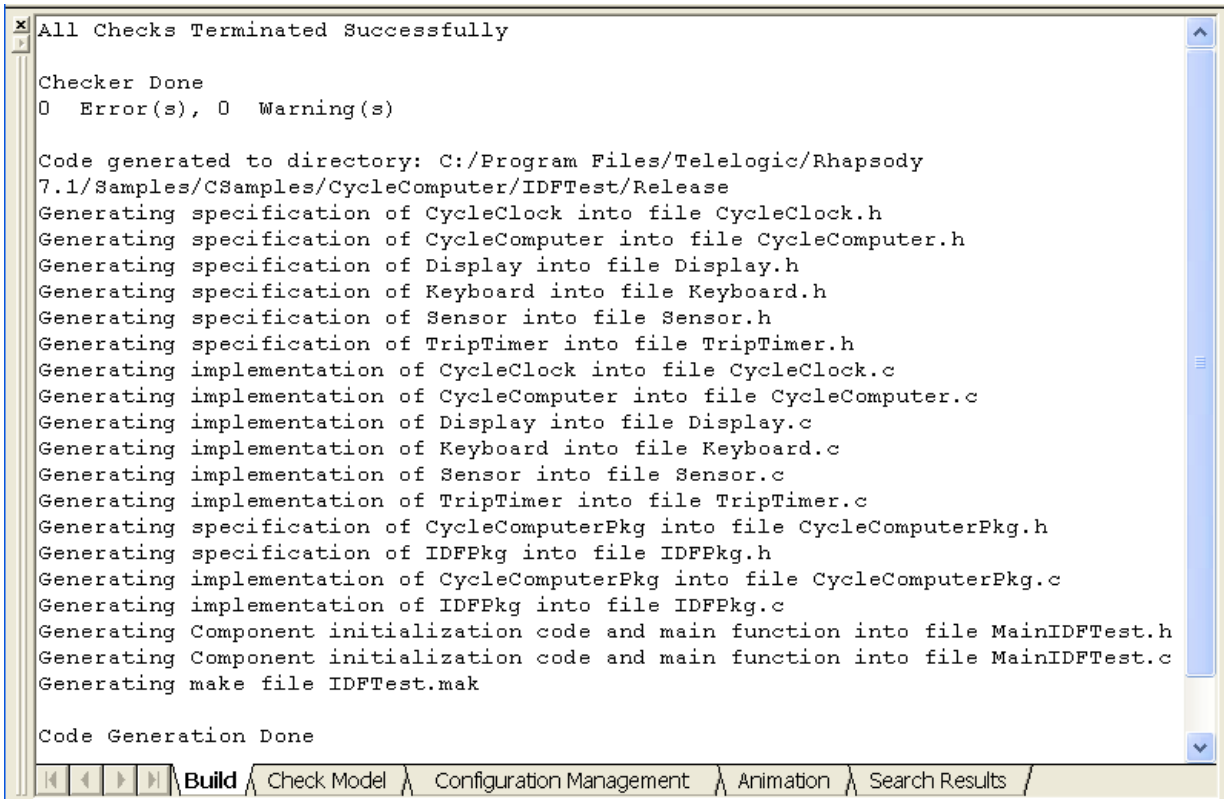
14. Select the **Settings** tab, from the pull-down menus at the bottom select **MicrosoftIDF** for the Environment and **Release** for the Build Set, as shown here. Click **OK**.



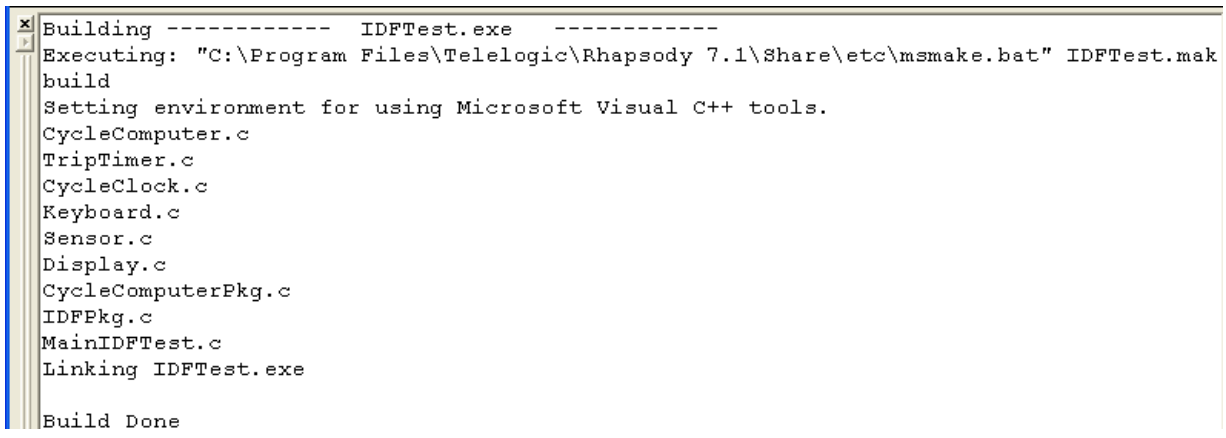
The browser tree for the new configuration should resemble this example.



15. Select the **Code > Generate >Release** options. The code generation messages should resemble these listed in the Build tab example.



16. To create the executable, select the **Code > Build IDFTest.exe (F7)** options. The build messages should resemble those in the following example:



17. To launch the TripTimer display, click the Run icon. The timer displays on a black background.

You may continue to experiment with this project to learn more about Rhapsody's IDF features. For example, if you do not want to use printf, add a define for the NO_PRINT macro.

Adapting the Framework for a Specific Target

To adapt the IDF to your target, follow these steps:

1. Create the new component and import the OSAL package from the Microsoft NT model provided (`Share\LangC\idf\Adapters\Microsoft\MicrosoftNT`). The following functions will have to be modified:
   - `RiCInitTimer`—sets up a periodic interrupt that calls the RiCTick operation every `RiC_MS_PER_TICK`.
   - `RiCExitCriticalRegion`—enables interrupts.
   - `RiCEnterCriticalRegion`—masks interrupts.
   - `RiCGetSystemTick`—returns system tick size.
   - `RiCSleep`—operation is called when there are no events to handle and sleep can be used until the next timeout or when an interrupt occurs.
2. Create `RiCOS`<target>`.c` and `RiCOS.h` files for the component and add the OSAL package as an element. Some examples of these filename formats are `$OMROOT\LangC\idf\RiCOSNT.c` and for similar target specific files, `$OMROOT\LangC\oxf`.
3. Create a directory called <target> under the `idf\Adapters` directory and set it as the output directory for the `RiCOS.h` file.
4. Include a new environment in the `siteC.prp` file for <Environment>IDF. Set up the makefile template, compiler flags, etc.
5. The following framework constants and types in the OSAL package must be set:
   - `RIC_MEMORY_ALLOCATION`—sets up the buffers used for the memory allocation.
   - `RIC_MAX_EVENTS`—maximum number of simultaneous events.
   - `RIC_MAX_TIMEOUTS`—maximum number of simultaneous timeouts.
   - `RIC_MS_PER_TICK`—periodic timeout in milliseconds.
   - `tRiCCriticalSection`—OS-specific type, which is used during critical section processing.
6. If you need `RiCString`, `RiCMap`, `RiCList`, and `RiCCollection`, then they can be added to the generic configuration scope and the idf library should be rebuilt. They are not included by default. To use `RiCString`, the user must `#include RiCString.h`.
7. Change the following properties in `C_CG::<Target>:`
   - `MakeFileName`—<target>`idf`

- `MakeFileContent`—change the name of the IDF library to <target>`idf$(LIB_EXT)`
- `CppCompileSwitches`- add the LangC/Adapters/<target> path.
- Add the following property:
  `Property ReactiveVtblKind Enum "OXF, IDF" "IDF"`

8. Add the profile `IDFProfile.sbs` to the model, as described in the previous section.

Note

If the message, "*CG MESSAGE: There are no classes in the component scope,"* displays while performing these steps, the message can be ignored.

Limitations of the IDF

While the IDF has a much smaller footprint than the standard framework provided with Rhapsody, the OXF, this size reduction brings with it the following limitations:

- The IDF is single-threaded and interrupt-driven.
- It is not possible to use the animation and tracing features with the IDF. All models, however, can be animated using the OXF.
- The IDF requires the use of the "flat statechart implementation" and "real-time model" options.
- The event queue and timer heaps are not dynamic. Maximum sizes must be set using the `RIC_MAX_EVENTS` and `RIC_MAX_TIMEOUTS` macro definitions.
- The property `CG::Events::BaseNumberOfInstances` must be set to a value greater than 0 to allow automatic allocation of memory from the memory pools. The actual number used will be ignored.
- In order to save RAM, the maximum number of consecutive null transitions has been greatly reduced - from 100 to 7.