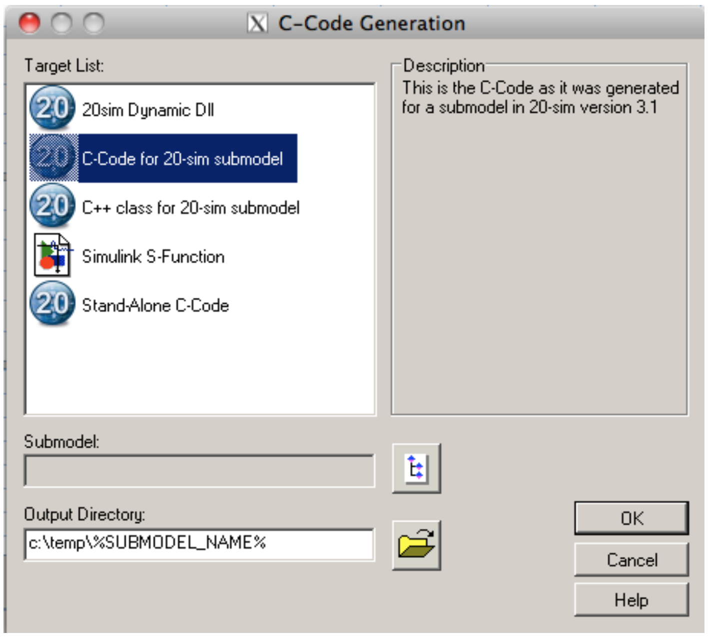
Code Generation in 20-sim

20-sim has a C-Code Generator which automatically converts a complete 20-sim model or submodel into C-Code.

To use the code generation feature in 20-sim go to 'Model->Start Simulator' to open a new simulator window. Then from the simulator window go to menu 'Tools-> Real Time Toolbox -> C-Code Generation' to display a window like one shown below.

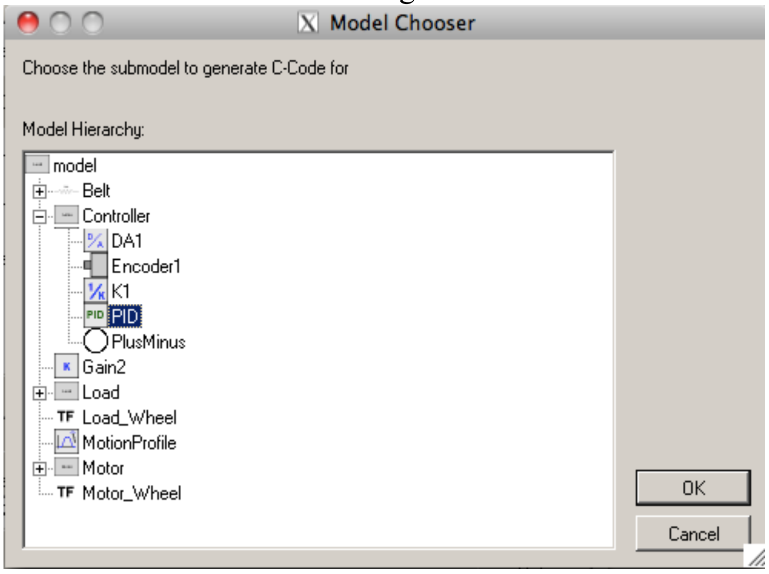


Description
This is the C-Code as it was generated for a submodel in 20-sim version 3.1

Output Directory: c:\temp\%SUBMODEL_NAME%

20 sim generates the code depending on the target platform selected from the list displayed in the Code Generation window. Each target can point to a set of template files that are used to create C-Code. The user can also generate a code for a specific submodel. For example the model for both Jiwy and Linix setups consists of the plant as well as the controllers and setup generators. Then the user can generate a c-code for the controller only.

For a C-code implementation of the PID controller chose the 'C-Cde for 20-sim submodel' option in the target and select the PID submodel under the Controller as a submodel. Select a destination to place the generated files and select 'OK' from code generation window.



It will generate a number of files with one xxmain.c. The file contains the implementation which includes the input, the output and the calculations in the submodel.

```
        /* initialize the submodel itself */
        XXInitializeSubmodel (u, y, xx_time);

        /* simple loop, the time is incremented by the integration method */
        while (xx_time < xx_finish_time)
        {
                /* call the submodel to calculate the output */
                XXCalculateSubmodel (u, y, xx_time);
        }

        /* perform the final calculations */
        XXTerminateSubmodel (u, y, xx_time);
```

This is the code that can replace the controller and generate periodic outputs during realization. But since the code uses a simulation based timer to implement the while() loop, think of a way to replace it with timer based implementation which takes into account the sample time used in the model.

The `XXCalculateSubmodel (u, y, xx_time)` function performs the computation used in the PID controller of the model. You can find the function implementation in `xxsubmod.c`. In the implementation of the function, `XXDiscreteStep ()` function calls another function `XXCalculateDynamic()` found in `xxmodel.c` which implements the same equations as the PID controller in the model.

```
void XXCalculateDynamic (void)
{
        /* factor = 1 / (sampletime + tauD * beta); */
        xx_V[1] = 1.0 / (xx_step_size + xx_P[1] * xx_P[2]);

        /* uD = factor * (((tauD * uD_previous) * beta + (tauD * kp) * (error -
error_previous)) + (sampletime * kp) * error); */
        xx_R[0] = xx_V[1] * (((xx_P[1] * xx_s[0]) * xx_P[2] + (xx_P[1] * xx_P[0]) *
(xx_R[1] - xx_s[1])) + (xx_step_size * xx_P[0]) * xx_R[1]);

        /* uI = uI_previous + (sampletime * uD) / tauI; */
        xx_R[2] = xx_s[2] + (xx_step_size * xx_R[0]) / xx_P[3];

        /* output = uI + uD; */
        xx_V[0] = xx_R[2] + xx_R[0];

        /* increment the step counter */
        xx_steps++;
}
```

Note: The input to the PID controller in the model is the position in radians so don't forget to change the encoder output to radians and give the setpoint in radians.

For the JIWY setup with two degree of freedom and two motors, you can generate the code for one controller and modify it to have four inputs and two outputs to have one controller for both the pan and tilt.