

Capsim® Block Generator

MINGW/MSYS Environment

Capsim ® Windows MINGW Version 6.0
(c) 1989-2016 Silicon Corporation, All Rights Reserved

Table of Contents

<i>Table of Contents</i>	2
<i>List of Figures</i>	3
<i>Introduction</i>	4
<i>Creating Working Directory for Custom Capsim</i>	4
<i>Creating a Custom Source Block</i>	7
<i>Examining the Source Code of the Generated Block</i>	11
<i>Adding the Block to Capsim</i>	16
<i>Creating a Process Block</i>	19
<i>Compilation Errors</i>	26
<i>Block Database Utility</i>	29

List of Figures

Figure 1 First Command to be Executed in New Directory	4
Figure 2 Console After Executing precapsim.sh Script	5
Figure 3 Folders and Files Created in Working Directory	6
Figure 4 Running Capsim	7
Figure 5 BLOCK Directory and Command to Bring Up Block Generation Tool	8
Figure 6 Block Generation Tool	9
Figure 7 Source Block Settings for xsource	10
Figure 8 New Block xsource.s Created	11
Figure 9 Source for xsource.s Block <BLOCK_NAME> Tag	12
Figure 10 Source Code xsource.s STATES and PARAMETERS with Amplitude Parameter	13
Figure 11 Source Code xsource.s MAIN_CODE	14
Figure 12 Code Change to Generate Linear Ramp	15
Figure 13 Make Creating C Code, Compiling and Linking Block into Capsim	16
Figure 14 Placing the xsource Block on the Workspace	17
Figure 15 Simulating the xsource Block	18
Figure 16 Block Generation Settings for xproces.s	19
Figure 17 Process Block Main Code	20
Figure 18 Process Block Modified C Code	21
Figure 19 Opening test_xsource	22
Figure 20 Moving the plot Block	23
Figure 21 Selecting Connection and Selecting Insert from Popup Menu	24
Figure 22 Insert Block List with xprocess Selected	24
Figure 23 Block Inserted	25
Figure 24 Running Simulation with xsource and xprocess	25
Figure 25 xprocess.s with Undefined Variable ZZZ Inserted	26
Figure 26 Compile with Error Messages using Make in WORK Directory	27
Figure 27 Compile with Error using make -f blocks.mak in BLOCKS Directory	28
Figure 28 Make in BLOCKS Directory with ZZZ Removed from xprocess.s No Errors	28
Figure 29 Block Management Utility	29

Introduction

This tutorial describes how to generate custom blocks using tools provided with Capsim. The tools use graphical user interfaces to simplify the generation of C embedded in XML source code for a variety of blocks.

Creating Working Directory for Custom Capsim

The first step in adding a new block to Capsim is to create a working directory to house the new blocks source code, C code and also any subroutines that may be used. Startup an MSYS console (go to c:\CAPSIM\CAPSIM_V6 and click on the MSYS shortcut). Type 'cd' to make sure you are in your home directory (usually /home/user_name). Create a new directory (we will create a directory called WORK):

```
mkdir WORK
```

Next type the command:

```
source $CAPSIM/TOOLS/precapsim.sh -l
```

See Figure 1.

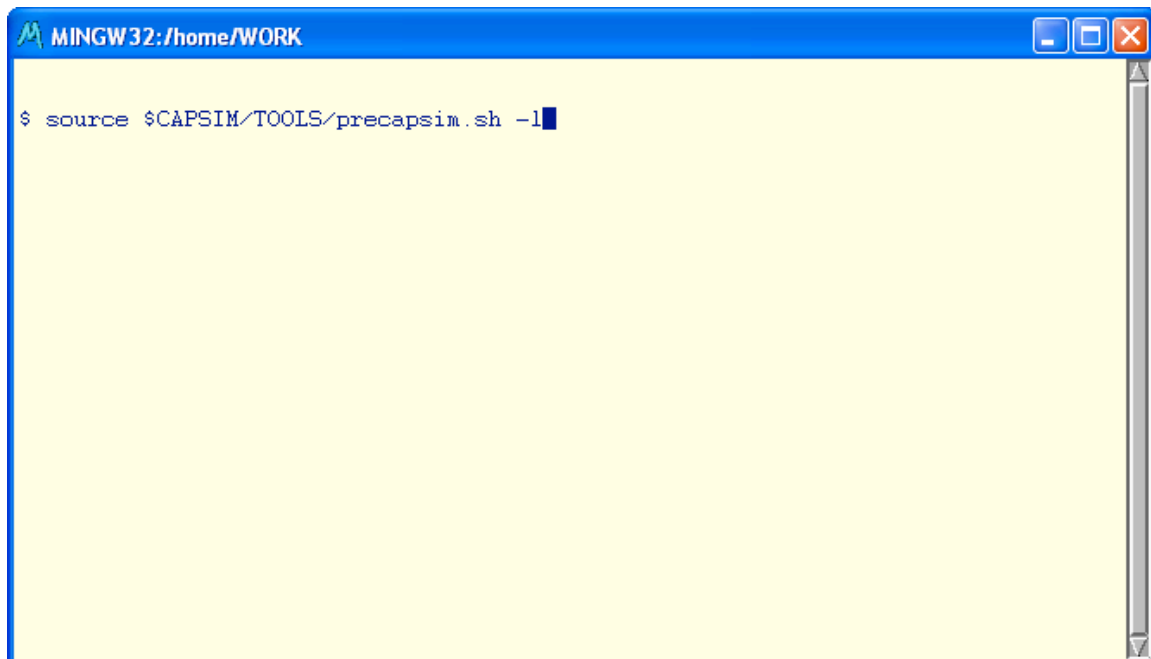
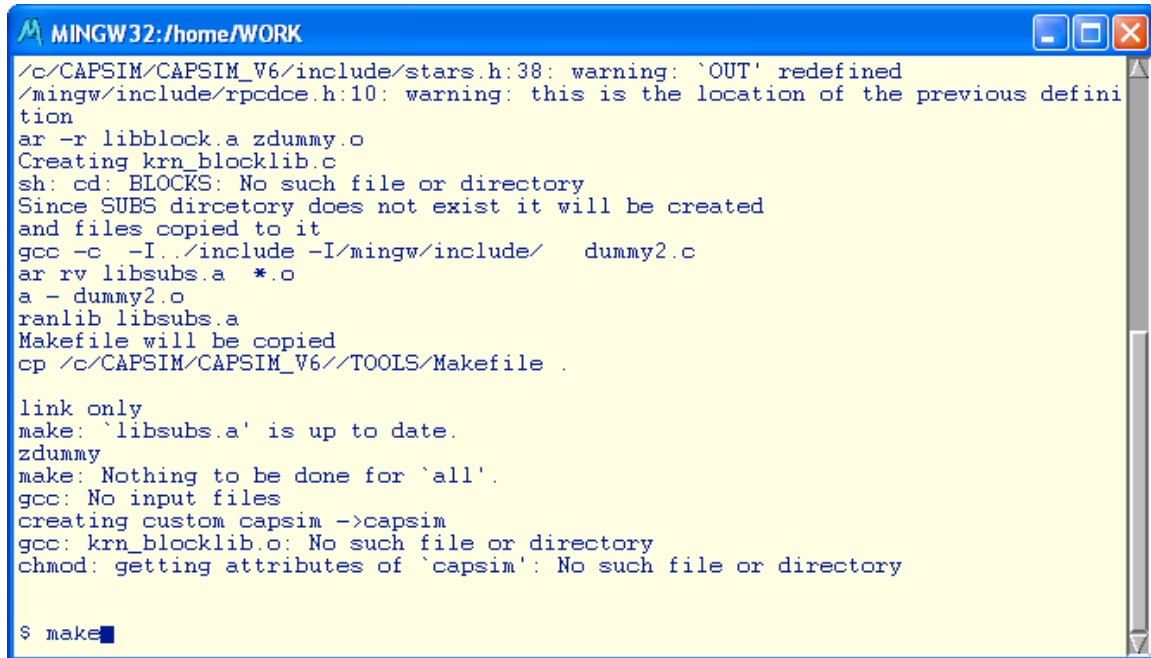


Figure 1 First Command to be Executed in New Directory

Figure 2 shows the result of executing the precapsim.sh script. After executing this command, type the make command:

make



```
MINGW32:/home/WORK
/c/CAPSIM/CAPSIM_V6/include/stars.h:38: warning: `OUT' redefined
/mingw/include/rpcdce.h:10: warning: this is the location of the previous defini
tion
ar -r libblock.a zdummy.o
Creating krn_blocklib.c
sh: cd: BLOCKS: No such file or directory
Since SUBS dirctory does not exist it will be created
and files copied to it
gcc -c -I../include -I/mingw/include/ dummy2.c
ar rv libsubs.a *.o
a - dummy2.o
ranlib libsubs.a
Makefile will be copied
cp /c/CAPSIM/CAPSIM_V6//TOOLS/Makefile .

link only
make: `libsubs.a' is up to date.
zdummy
make: Nothing to be done for `all'.
gcc: No input files
creating custom capsim ->capsim
gcc: krn_blocklib.o: No such file or directory
chmod: getting attributes of `capsim': No such file or directory

$ make
```

Figure 2 Console After Executing precapsim.sh Script

After executing the precapsim,.sh script and make type 'ls' as shown in Figure 3. Notice that two new directories BLOCKS and SUBS have been created. The also a new CAPSIM executable *capsim.exe* is created. Notice also the *Makefile*. When you want to update CAPSIM after you add a block or subroutine, the *Makefile* is used to create a new *capsim.exe* executable. To bring up CAPSIM type:

capsim

Figure 4 shows the Capsim workspace and console.

A screenshot of a MINGW32 terminal window. The title bar shows 'MINGW32:/home/WORK'. The terminal content shows a command prompt '\$ ls' followed by the output 'BLOCKS Makefile SUBS capsim.exe krn_blocklib.o'. A second prompt '\$' with a cursor is visible on the next line.

```
MINGW32:/home/WORK
$ ls
BLOCKS Makefile SUBS capsim.exe krn_blocklib.o
$ █
```

Figure 3 Folders and Files Created in Working Directory

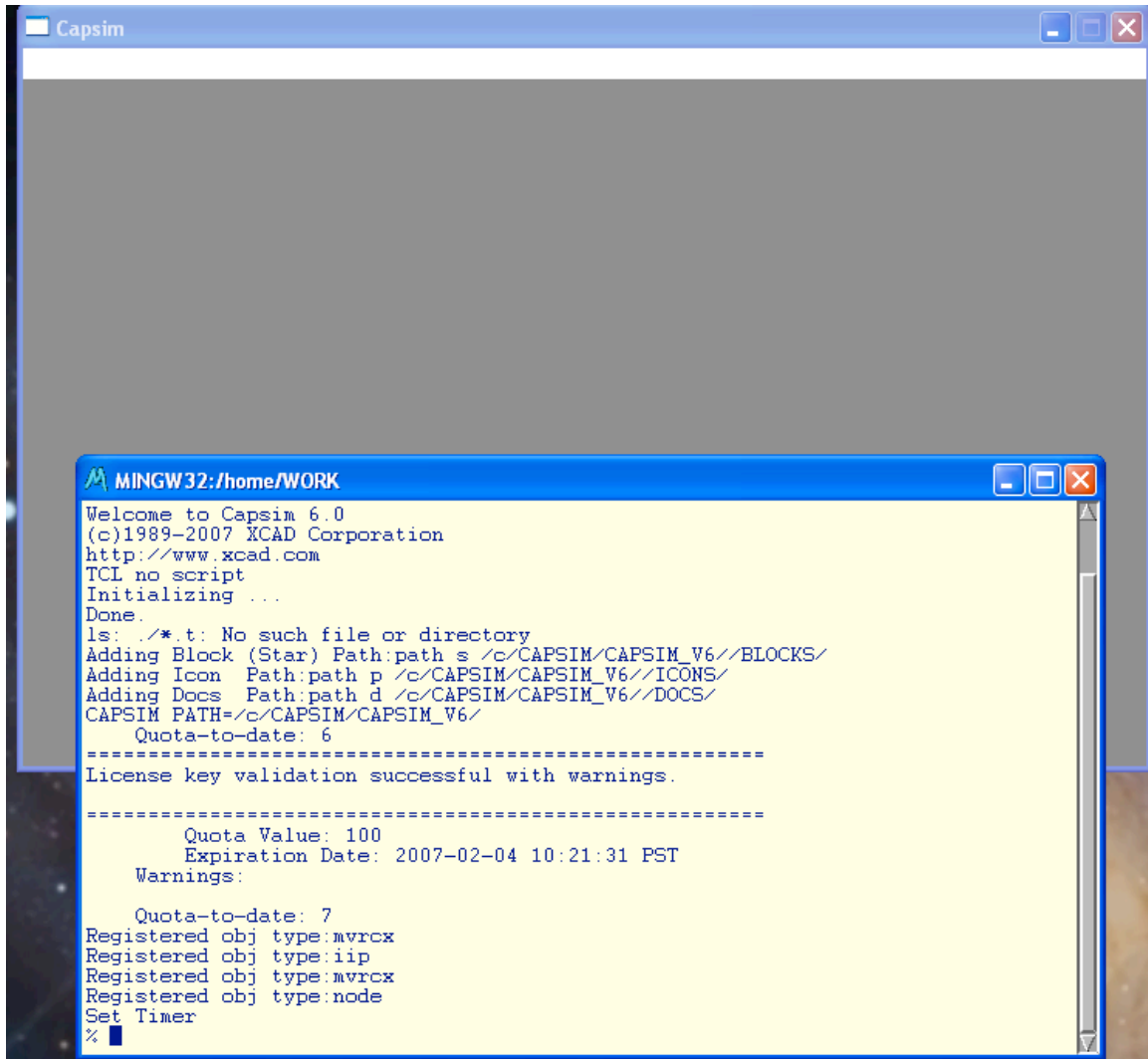


Figure 4 Running Capsim

Notice the Licensing information and the registration of types for the built in TCL interpreter. Always hit RETURN in the console window to kick start the graphics user interface. (Always place the console such that you can see the messages).

Creating a Custom Source Block

In this section we will create a source block and add it to CAPSIM. Change directories to the BLOCK directory by typing

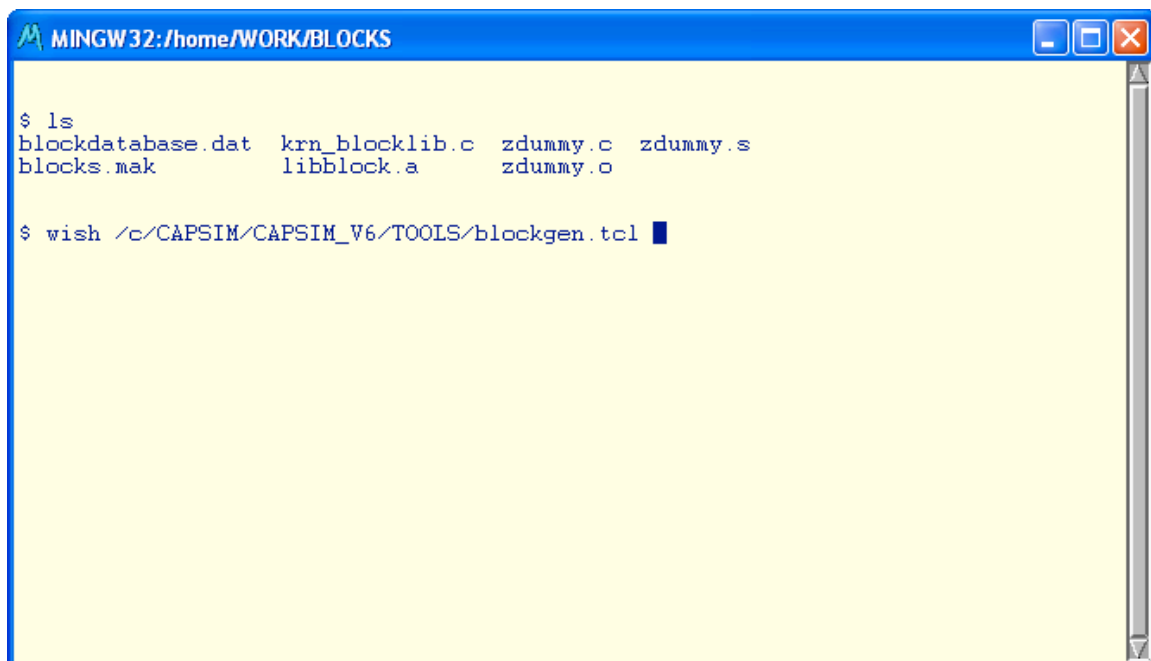
```
cd BLOCKS
```

Type 'ls' as shown in Figure 5. Note the files in the block directory. There is a dummy block called `zdummy.s`, a make file for blocks, `blocks.mak`, a block database of all blocks included in `capsim`, `blockdatabase.dat`. Finally there is the C code `krn_blocklib.c` which is linked into CAPSIM.

To start the block generation tool type the following command as shown in Figure 5:

```
wish $CAPSIM/TOOLS/blockgen.tcl
```

Note that `wish` is the command to invoke TK/TCL.



```
MINGW32:/home/WORK/BLOCKS
$ ls
blockdatabase.dat  krn_blocklib.c  zdummy.c  zdummy.s
blocks.mak         libblock.a      zdummy.o

$ wish /c/CAPSIM/CAPSIM_V6/TOOLS/blockgen.tcl
```

Figure 5 BLOCK Directory and Command to Bring Up Block Generation Tool

The Block Generation User Interface shown in Figure 6 appears. There are two windows. The key window is the `blockgen` window (to the right). In this window you select the type of block to generate and the type of input/output buffer (float, int, fixed point, complex and image). The second window is for specifying parameters.

We will create a source block called `xsource`. It will have a floating point buffer and a parameter called `amplitude`. Select "Source" for the Block Type Radio Button. Select float for the Buffer Type. Type in the blocks name in the name field (`xsource`). In the parameter window type `Amplitude` for the Parameter Prompt. Type `100.0` for the Default Parameter Value. For the Parameter Name type `amplitude`. Click on the Add Parameter

button to add the parameter to the list. If you make a mistake you can select the parameter and delete it.

After completing the above steps (double check against Figure 7) click on the Generate button in the *blockgen* window. A confirmation for the the block generation will appear. Click Okay.

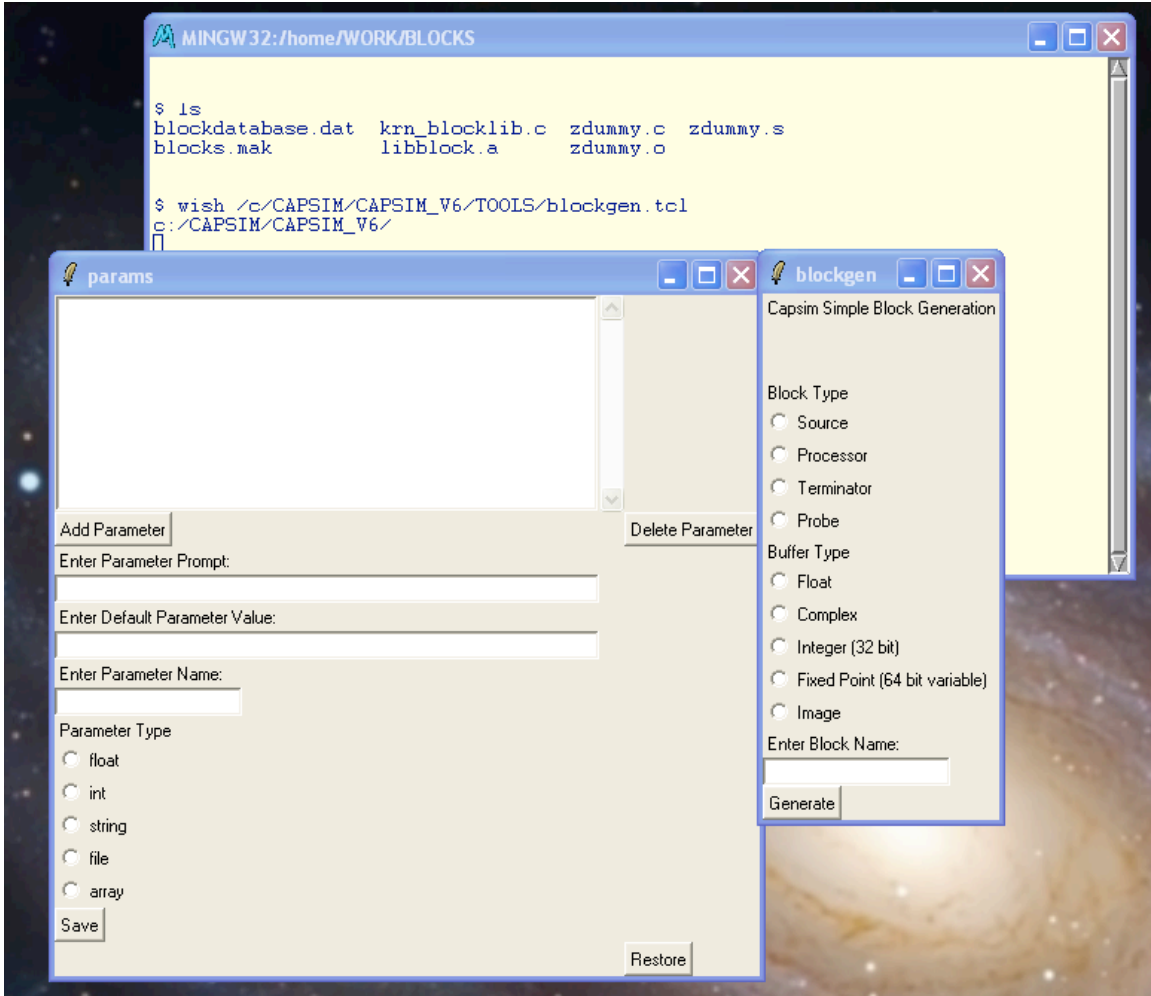


Figure 6 Block Generation Tool

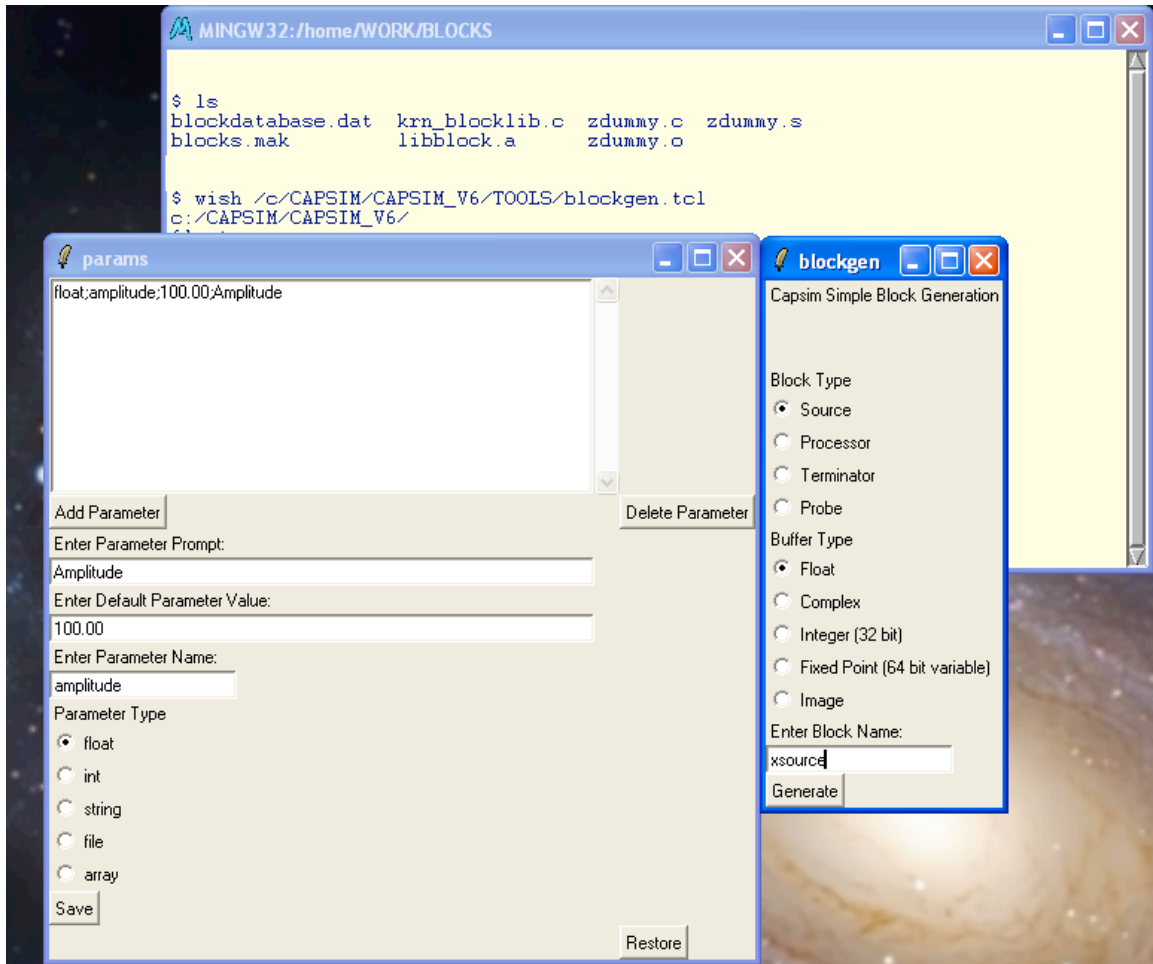
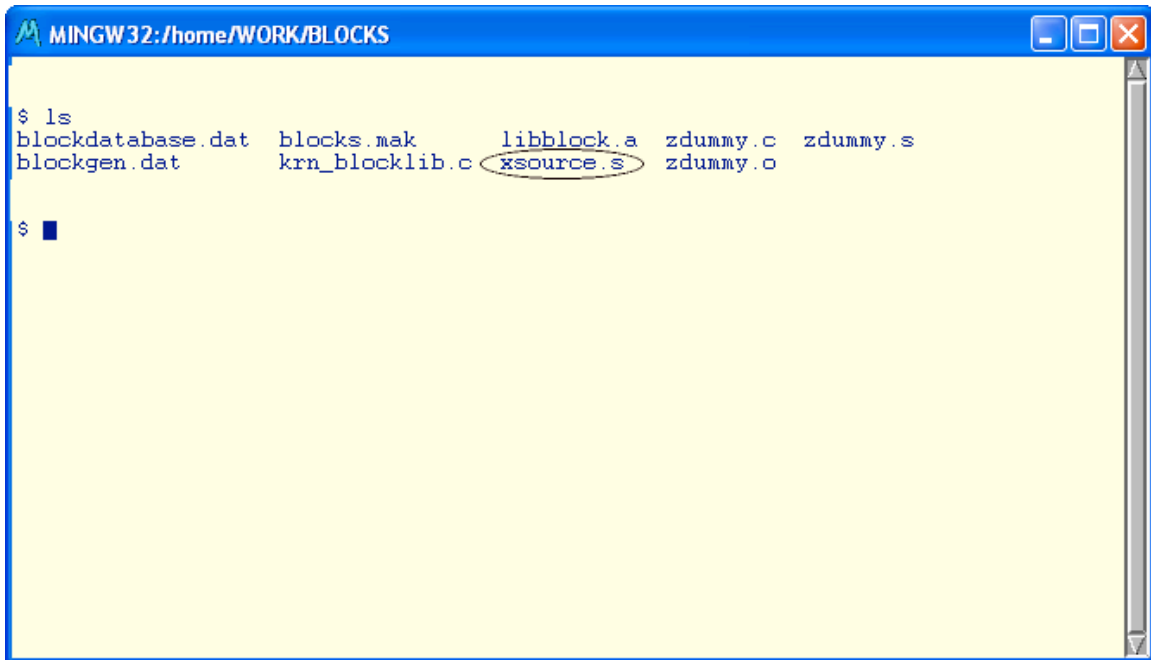


Figure 7 Source Block Settings for xsource

Go ahead and quit the Block Generation Tool by closing the *blockgen* window. Type 'ls' in the BLOCKS directory.



```
MINGW32:/home/WORK/BLOCKS
$ ls
blockdatabase.dat  blocks.mak  libblock.a  zdummy.c  zdummy.s
blockgen.dat       krn_blocklib.c  xsource.s  zdummy.o
```

Figure 8 New Block *xsource.s* Created

Examining the Source Code of the Generated Block

In the following we will examine the source code of the generated BLOCK *xsource.s*. We will modify it so that it produces a linear ramp depending on the amplitude.

```

<BLOCK>
<LICENSE>
/*
 * (c) 2006 AUTHOR_NAME
 */

</LICENSE>
<BLOCK_NAME>
xsource
</BLOCK_NAME>
|

<DESC_SHORT>
Add short description here. Will appear in HTML documentation.
</DESC_SHORT>

<COMMENTS>
<![CDATA[

/*
 *          xsource()
 *
 * Generated by blockgen
 *
 * Floating point source
 * Notes:
 *   This is a block generated from the floating point source template
 *   It will output numberOfSamples of samples defined by a parameter.
 *   It outputs the floating point constant 1.0
 *
 *
 * Programmer:
 * Date:
 * Modified:
 *
 */
]]>
</COMMENTS>

```

Figure 9 Source for xsource.s Block <BLOCK_NAME> Tag

```

<DECLARATIONS>
    int i,j;
</DECLARATIONS>

<STATES>
    <STATE>
        <TYPE> int </TYPE>
        <NAME> done </NAME>
        <VALUE> FALSE </VALUE>
    </STATE>
    <STATE>
        <TYPE> int </TYPE>
        <NAME> sampleCount </NAME>
        <VALUE> 0 </VALUE>
    </STATE>
</STATES>

<PARAMETERS>
    <PARAM>
        <DEF> Amplitude </DEF>
        <TYPE> float </TYPE>
        <NAME> amplitude </NAME>
        <VALUE> 100.00 </VALUE>
    </PARAM>
    <PARAM>
        <DEF> Total number of samples to output </DEF>
        <TYPE> int </TYPE>
        <NAME> numberOfSamples </NAME>
        <VALUE> 1000 </VALUE>
    </PARAM>
</PARAMETERS>

```

Figure 10 Source Code *xsource.s* STATES and PARAMETERS with Amplitude Parameter

Figure 10 shows the PARAMETER Tag. Note the inclusion of the Amplitude parameter. Also the numberOfSamples parameter is automatically added by the block generator.

```

<OUTPUT_BUFFERS>
  <BUFFER>
    <TYPE> float </TYPE>
    <NAME> x </NAME>
  </BUFFER>
</OUTPUT_BUFFERS>

<INIT_CODE>
<![CDATA[
]]>
</INIT_CODE>

<MAIN_CODE>
<![CDATA[ |
  if(done)return(0);
  /*
   * output a maximum of NUMBER_SAMPLES_PER_VISIT samples to output buffer(s)
   */
  for(i=0; i < NUMBER_SAMPLES_PER_VISIT; ++i) {
    sampleCount++;
    if(sampleCount == numberOfSamples) {
      done=TRUE;
      return(0);
    }

    /*
     * ready output buffer for sample
     * check for overflow
     */
    if(IT_OUT(0)) {
      KrnOverflow("xsource",0);
      return(99);
    }
    /*
     * output the sample
     */
    x(0)=1.0;
  }
]]>
</MAIN_CODE>

```

Figure 11 Source Code *xsource.s* MAIN_CODE

In the MAIN_CODE tag note that the output buffer x(0) is set to 1.0.

```

  /*
   * Output the sample
   */
  x(0)=1.0;

```

We will change this to the following (use your favorite editor):

```
    /*
    * Output the sample
    */
    x(0)=amplitude*sampleCount/numberOfSamples;
```

See Figure 12.

```
<OUTPUT_BUFFERS>
  <BUFFER>
    <TYPE> float </TYPE>
    <NAME> x </NAME>
  </BUFFER>
</OUTPUT_BUFFERS>

<INIT_CODE>
<![CDATA[
]]>
</INIT_CODE>

<MAIN_CODE>
<![CDATA[
  if(done)return(0);
  /*
  * output a maximum of NUMBER_SAMPLES_PER_VISIT samples to output buffer(s)
  */
  for(i=0; i < NUMBER_SAMPLES_PER_VISIT; ++i) {
    sampleCount++;
    if(sampleCount == numberOfSamples) {
      done=TRUE;
      return(0);
    }

    /*
    * ready output buffer for sample
    * check for overflow
    */
    if(IT_OUT(0)) {
      KrnOverflow("xsource",0);
      return(99);
    }
    /*
    * output the sample
    */
    x(0)=amplitude*sampleCount/numberOfSamples;

  }
]]>
</MAIN_CODE>
```

Figure 12 Code Change to Generate Linear Ramp

In the code, *sampleCount* is a State Variable that keeps track of the number of samples generated. The variable *numberOfSamples* is a parameter that specifies the number of samples to generate. The variable *amplitude* is the parameter that we specified in the block generation tool. (In the generated C code these are actually defined MACROS that point to variables in data structures, i.e. States and Parameters). Save the changes.

Adding the Block to Capsim

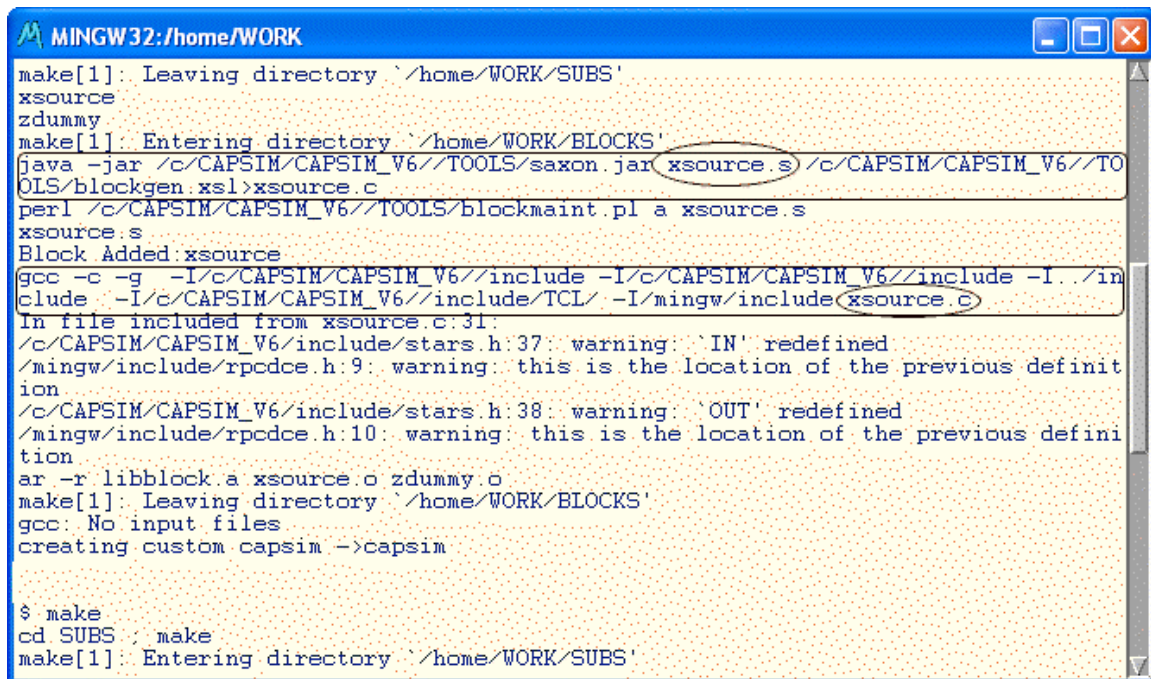
At this point, change directories back to the top (the directory WORK) by typing

```
cd ..
```

Next type:

```
make
```

The block code is converted from XML to C code by Java. The C code is compiled. The block is added to the *libblock.a* archive and linked into Capsim. A new executable *capsim.exe* is created. If a new block is added, always execute make again. Otherwise the block will not appear in the block list. This is not necessary for updating existing blocks. Just type *make* once.



```
MINGW32:/home/WORK
make[1]: Leaving directory `/home/WORK/SUBS'
xsource
zdummy
make[1]: Entering directory `/home/WORK/BLOCKS'
java -jar /c/CAPSIM/CAPSIM_V6//TOOLS/saxon.jar xsource.s /c/CAPSIM/CAPSIM_V6//TOOLS/blockgen.xsl>xsource.c
perl /c/CAPSIM/CAPSIM_V6//TOOLS/blockmaint.pl a xsource.s
xsource.s
Block Added:xsource
gcc -c -g -I/c/CAPSIM/CAPSIM_V6//include -I/c/CAPSIM/CAPSIM_V6//include -I../include -I/c/CAPSIM/CAPSIM_V6//include/TCL/ -I/mingw/include xsource.c
In file included from xsource.c:31:
/c/CAPSIM/CAPSIM_V6/include/stars.h:37: warning: `IN' redefined
/mingw/include/rpcdce.h:9: warning: this is the location of the previous definition
/c/CAPSIM/CAPSIM_V6/include/stars.h:38: warning: `OUT' redefined
/mingw/include/rpcdce.h:10: warning: this is the location of the previous definition
ar -r libblock.a xsource.o zdummy.o
make[1]: Leaving directory `/home/WORK/BLOCKS'
gcc: No input files
creating custom capsim ->capsim

$ make
cd SUBS : make
make[1]: Entering directory `/home/WORK/SUBS'
```

Figure 13 Make Creating C Code, Compiling and Linking Block into Capsim

Hopefully all went well. If you see any errors during compilation there are limited to the code you changed. Focus on that code, fix it in the BLOCKS directory and type make in the WORK directory.

It is time to run CAPSIM with the incorporated *xsource* block. Type capsim in the WORK directory. This will bring up CAPSIM (remember type RETURN in the console window to kick start).

From the File menu select “New” to create a new blank workspace.

From the Blocks menu select “Blocks ...”. The list of blocks will appear. Scroll down (to the bottom) and select *xsource*. (If you can’t find *xsource*, then exit CAPSIM and type make again. Look for any errors and fix). Place the *xsource* block on the workspace as shown in Figure 14.

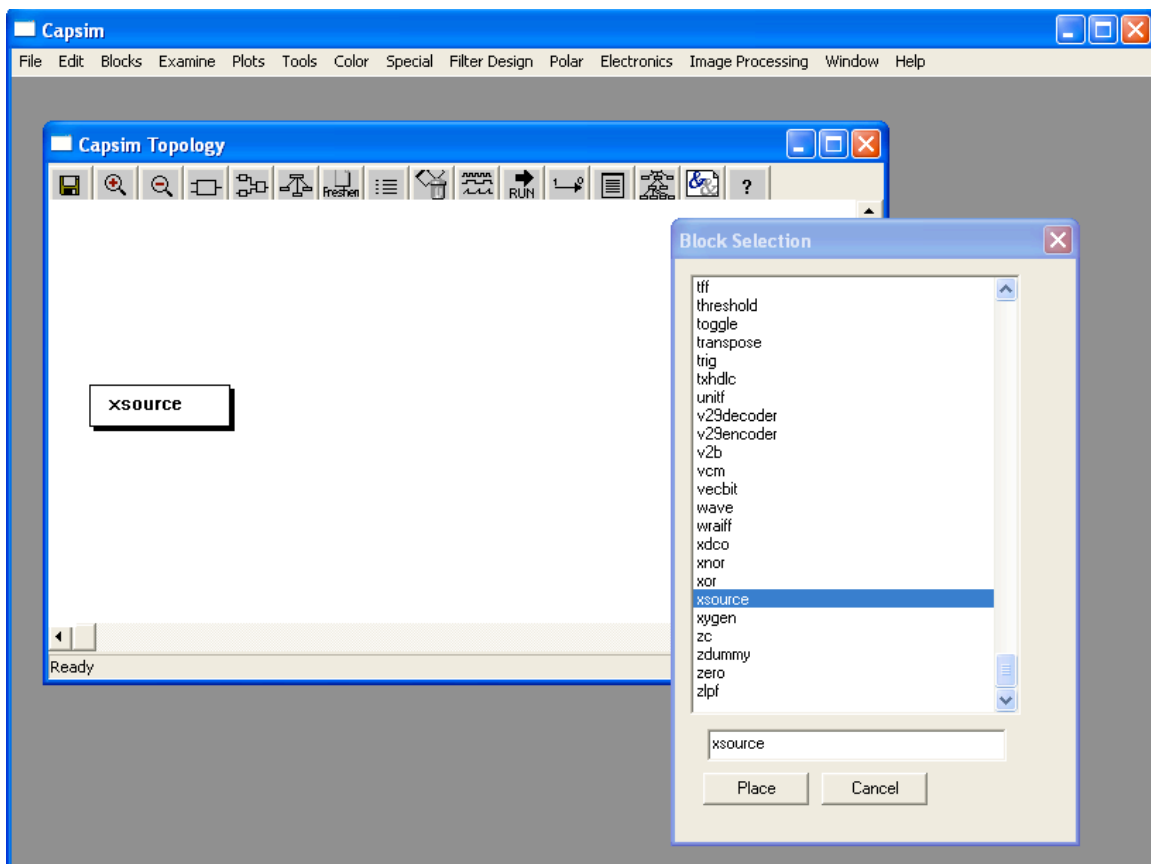


Figure 14 Placing the xsource Block on the Workspace

Next select the plot block from the list and place it on the workspace. Connect the xsource block to the plot block. Save the topology (name it test_xsource). Run the simulation. A plot should appear as shown in Figure 15.

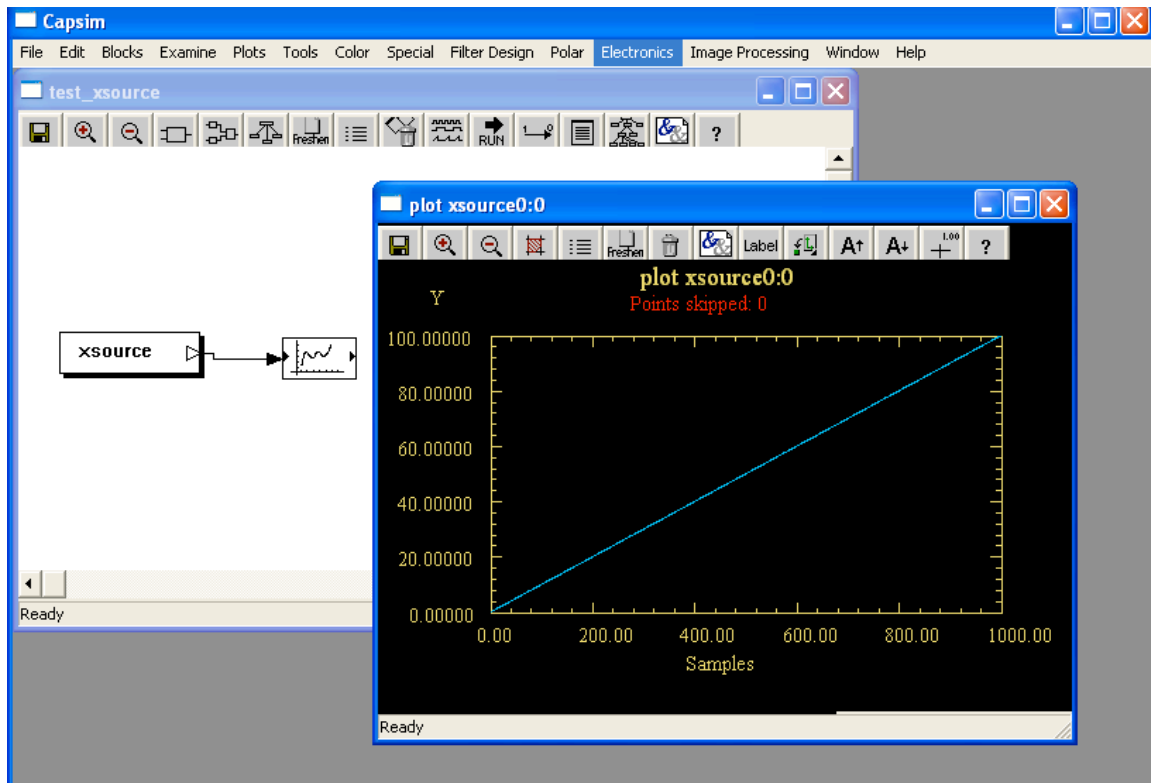


Figure 15 Simulating the *xsource* Block

Creating a Process Block

In this section we will generate a Processing Block. That is, a block that inputs samples, processes them, and outputs the processed samples. To generate the Processor block bringup the block code generation tool by changing directories to the BLOCKS directory. Then type

```
wish $CAPSIM/TOOLS/blockgen.tcl
```

The block generator window is displayed. Set the Block Type to “Processor”. Set the Buffer Type to “float”. Set the Block Name to “xprocess”. See Figure 16.

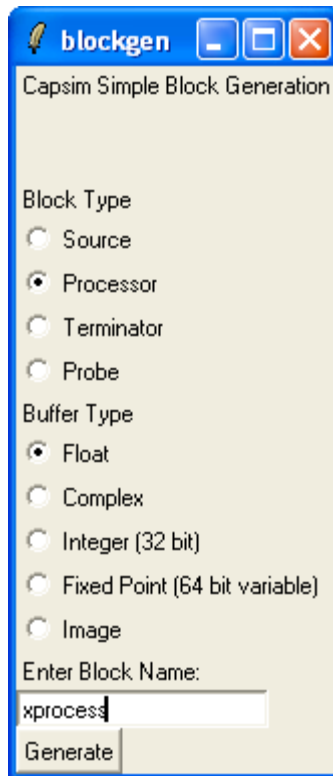


Figure 16 Block Generation Settings for xprocess.s

The file “xprocess.s” will be generated with code for a processor block. Edit this file and review the code. In Figure 17, we show the code for the MAIN_CODE tag. Note that the output sample is set to the value of the input sample. We will change the code so that the *xprocess.s* block squares its input sample and outputs the squared values. Make the

changes shown in Figure 18, save the file and cd to the WORK directory (cd ..) and type make TWICE (since we are adding a new block).

```
<MAIN_CODE>
<![CDATA[

while(IT_IN(0)) {
    sample=x(0);

    /*
     * ready output buffer for sample
     * check for overflow
     */
    if(IT_OUT(0)) {
        KrnOverflow("xprocess",0);
        return(99);
    }
    /*
     * output the sample
     */
    y(0)=sample;

}

]]>
</MAIN_CODE>

<WRAPUP_CODE>
<![CDATA[

]]>
</WRAPUP_CODE> |

</BLOCK>
```

Figure 17 Process Block Main Code

```

<MAIN_CODE>
<![CDATA[

while(IT_IN(0)) {
    sample=x(0);

    /*
     * ready output buffer for sample
     * check for overflow
     */
    if(IT_OUT(0)) {
        KrnOverflow("xprocess",0);
        return(99);
    }
    /*
     * output the sample
     */
    y(0)=sample*sample/10.0;

}

]]>
</MAIN_CODE>

<WRAPUP_CODE>
<![CDATA[

]]>
</WRAPUP_CODE>

</BLOCK>

```

Figure 18 Process Block Modified C Code

After capsim has been successfully built, run CAPSIM and open the previous topology “test_xsource.t). Figure 19 shows the topology. We will insert the *xprocess* block between the *xsource* and *plot* blocks. To do this, we need to move the *plot* block to the right to create some room. See Figure 20.

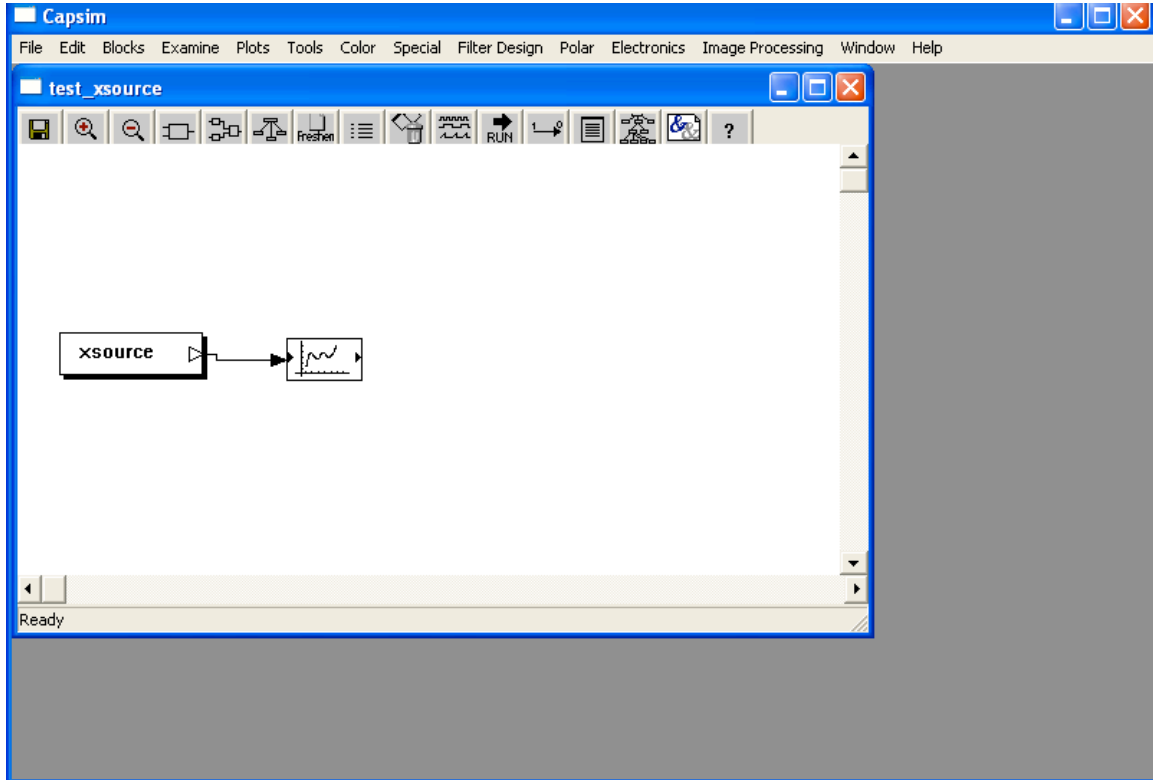


Figure 19 Opening *test_xsource*

Next click on the connection between the two blocks. The Connection Popup will appear. Select Insert. A sub menu will appear (shows Star for Block and Galaxy for Hblock). Select Star(Block). A list of blocks will appear. Scroll down to the xprocess block. (If it does not appear you need to type make again and look for possible errors in compilation). Click on the Place button, The xprocess block will be inserted between them as show in Figure 23. Run the simulation. You should get a plot of a parabola (square of the linear curve with *xsource* alone).

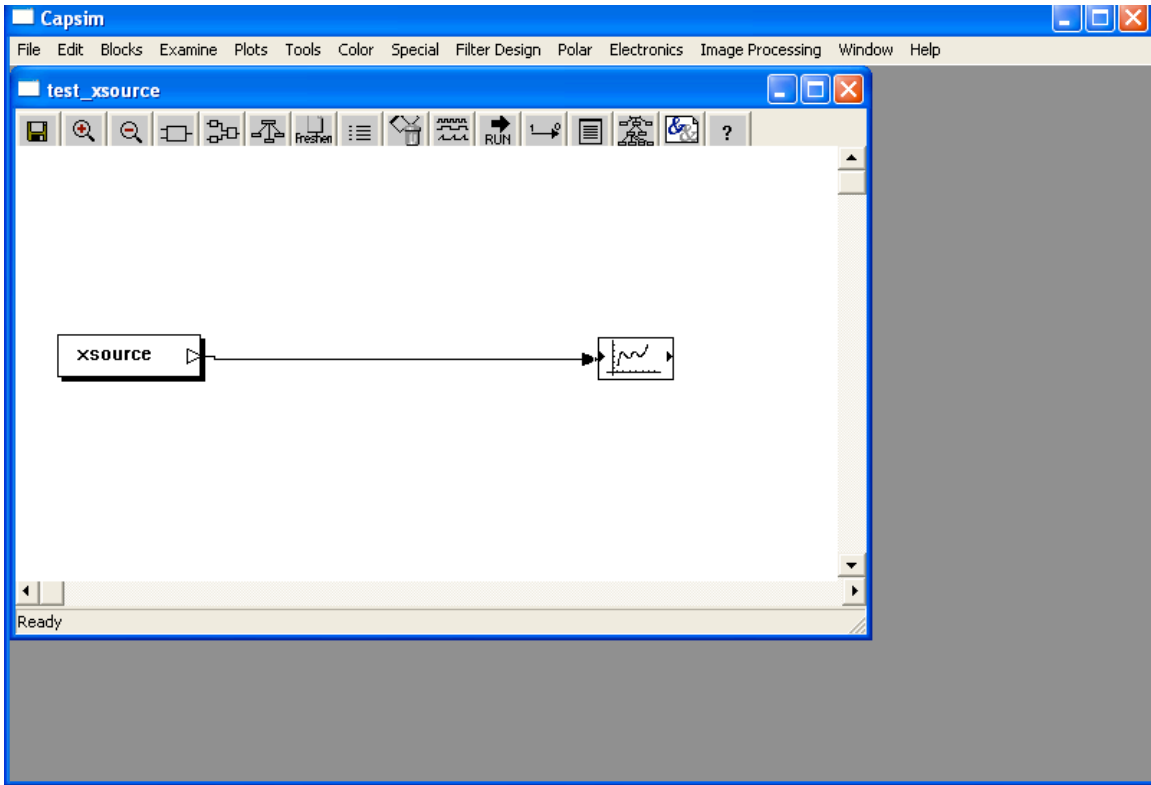


Figure 20 Moving the plot Block

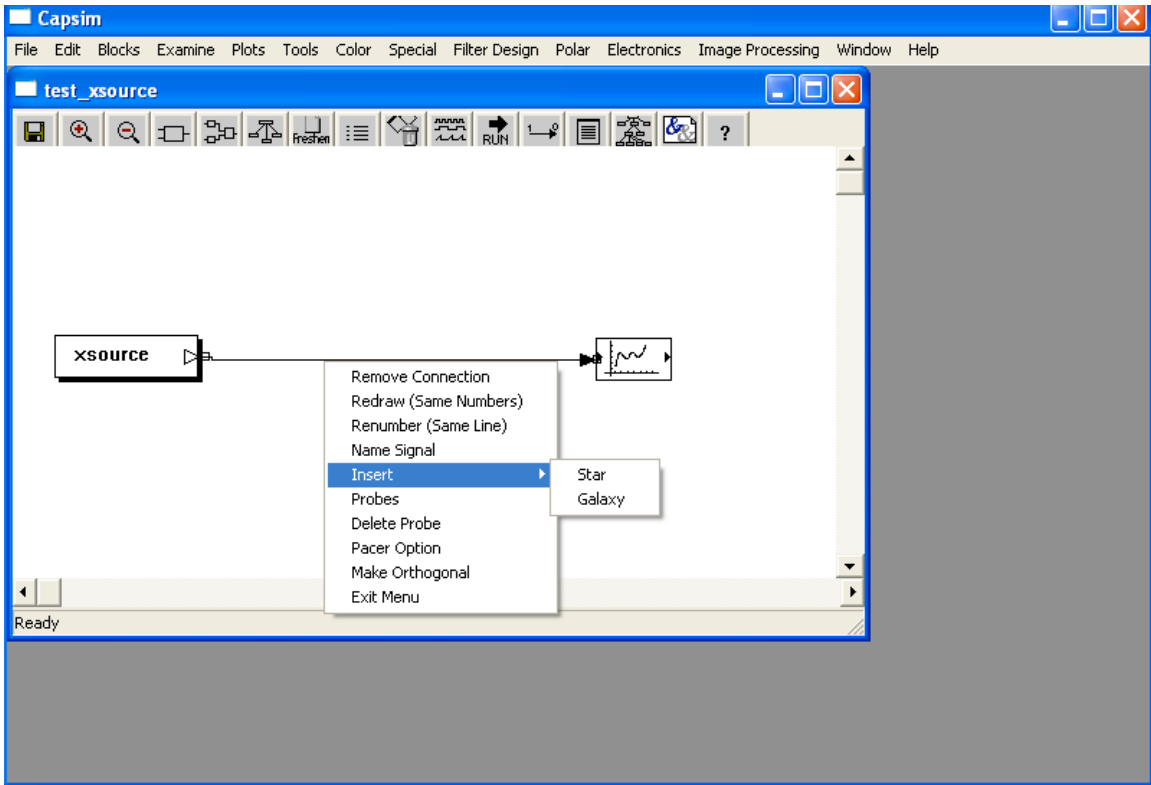


Figure 21 Selecting Connection and Selecting Insert from Popup Menu

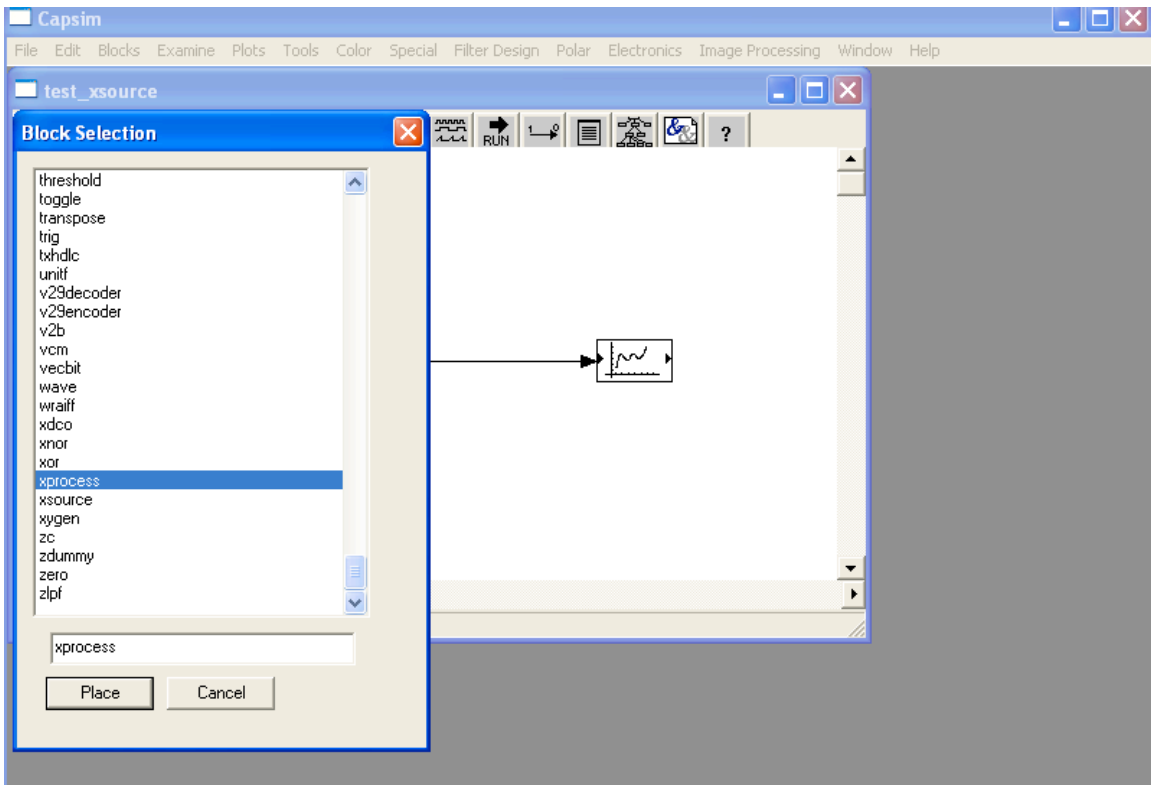


Figure 22 Insert Block List with *xprocess* Selected

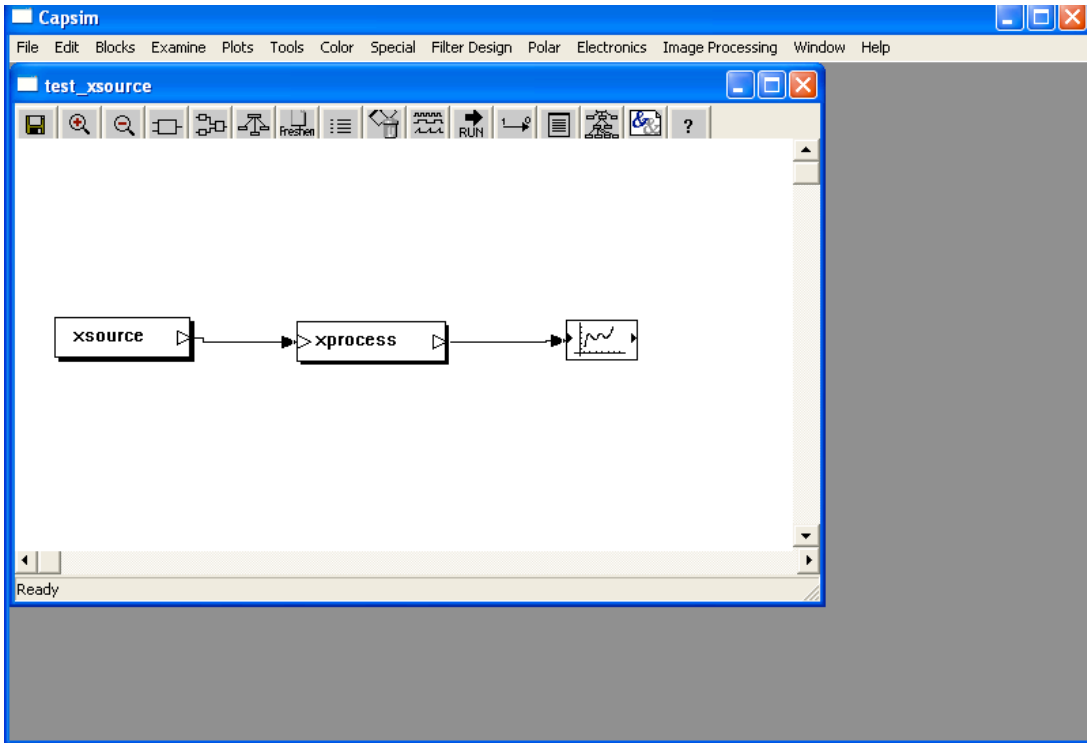


Figure 23 Block Inserted

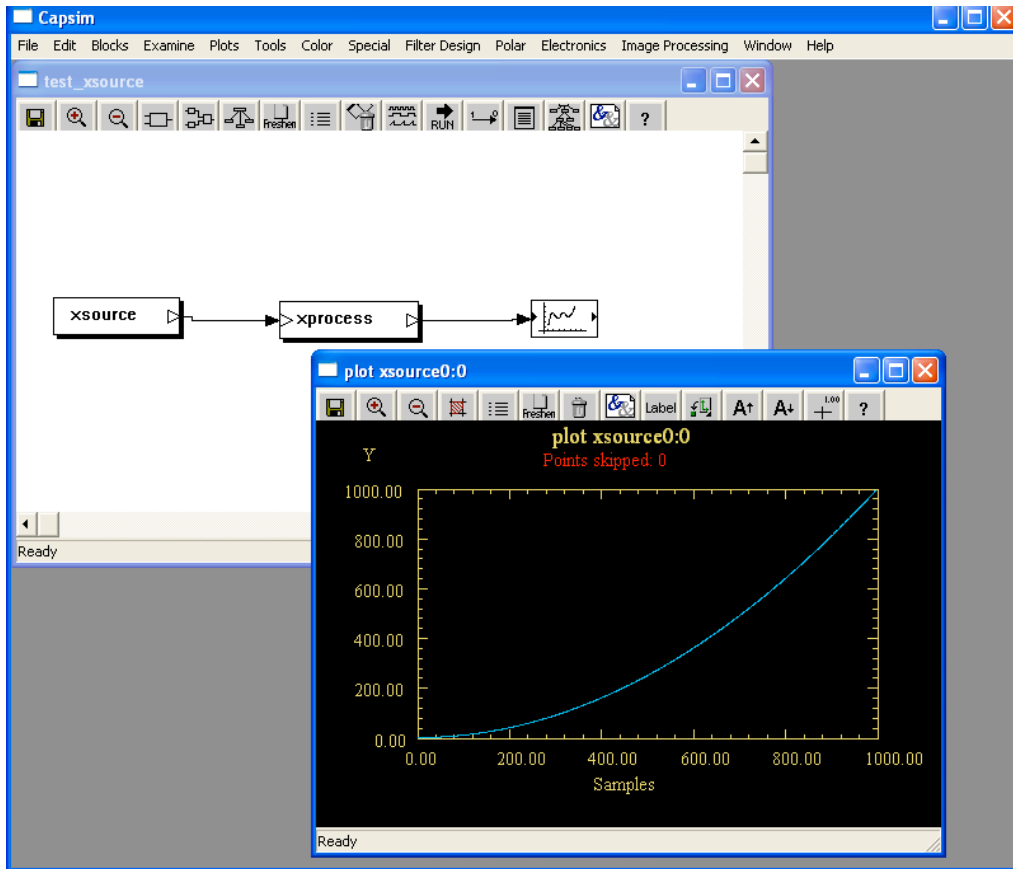


Figure 24 Running Simulation with *xsource* and *xprocess*

Compilation Errors

In this section we will add an undefined variable *ZZZ* to the *MAIN_CODE* section of *xprocess.s* and examine the compilation errors that result. Figure 25 shows the *MAIN_CODE* of *xprocess.s* with the undefined variable *ZZZ* inserted. If we *cd ..* to *WORK* and type *make* in the *WORK* directory we get compilation errors as shown in Figure 26.

```
while(IT_IN(0)) {
    sample=x(0);

    /*
     * ready output buffer for sample
     * check for overflow
     */
    if(IT_OUT(0)) {
        KrnOverflow("xprocess",0);
        return(99);
    }
    /*
     * output the sample
     */
    y(0)=sample*sample/10.0*ZZZ;
|
}

]]>
</MAIN_CODE>

<WRAPUP_CODE>
<![CDATA[

]]>
</WRAPUP_CODE>

</BLOCK>
```

Figure 25 *xprocess.s* with Undefined Variable *ZZZ* Inserted

```
MINGW32:/home/WORK
xprocess
xsource
zdummy
make[1]: Entering directory `/home/WORK/BLOCKS'
java -jar /c/CAPSIM/CAPSIM_V6//TOOLS/saxon.jar xprocess.s /c/CAPSIM/CAPSIM_V6//T
TOOLS/blockgen.xsl>xprocess.c
perl /c/CAPSIM/CAPSIM_V6//TOOLS/blockmaint.pl a xprocess.s
xprocess.s
BLOCK_ALREADY_EXISTS:xprocess
gcc -c -g -I/c/CAPSIM/CAPSIM_V6//include -I/c/CAPSIM/CAPSIM_V6//include -I../in
clude -I/c/CAPSIM/CAPSIM_V6//include/TCL/ -I/mingw/include xprocess.c
In file included from xprocess.c:31:
/c/CAPSIM/CAPSIM_V6/include/stars.h:37: warning: `IN' redefined
/mingw/include/rpcdce.h:9: warning: this is the location of the previous definit
ion
/c/CAPSIM/CAPSIM_V6/include/stars.h:38: warning: `OUT' redefined
/mingw/include/rpcdce.h:10: warning: this is the location of the previous defini
tion
xprocess.c: In function `xprocess':
xprocess.c:192: `ZZZ' undeclared (first use in this function)
xprocess.c:192: (Each undeclared identifier is reported only once
xprocess.c:192: for each function it appears in.)
make[1]: *** [xprocess.o] Error 1
make[1]: Leaving directory `/home/WORK/BLOCKS'
gcc: No input files
creating custom capsim ->capsim
$
```

Figure 26 Compile with Error Messages using Make in WORK Directory

On the other hand, if type `make -f blocks.mak` in the BLOCKS directory we get Figure 27 . Thus it is straight forward to detect and fix compilation errors in the BLOCKS directory. Figure 28 shows the `make -f blocks.mak` with the ZZZ undefined variable removed (problem fixed). After all updates to blocks type make in the WORK directory.

```
MINGW32:/home/WORK/BLOCKS
$ make -f blocks.mak
gcc -c -g -I/c/CAPSIM/CAPSIM_V6//include -I/c/CAPSIM/CAPSIM_V6//include -I../in
clude -I/c/CAPSIM/CAPSIM_V6//include/TCL/ -I/mingw/include xprocess.c
In file included from xprocess.c:31:
/c/CAPSIM/CAPSIM_V6/include/stars.h:37: warning: `IN' redefined
/mingw/include/rpcdce.h:9: warning: this is the location of the previous definit
ion
/c/CAPSIM/CAPSIM_V6/include/stars.h:38: warning: `OUT' redefined
/mingw/include/rpcdce.h:10: warning: this is the location of the previous defini
tion
xprocess.c: In function `xprocess':
xprocess.c:192: `ZZZ' undeclared (first use in this function)
xprocess.c:192: (Each undeclared identifier is reported only once
xprocess.c:192: for each function it appears in.)
make: *** [xprocess.o] Error 1

$ █
```

Figure 27 Compile with Error using *make -f blocks.mak* in BLOCKS Directory

```
MINGW32:/home/WORK/BLOCKS
$ make -f blocks.mak
java -jar /c/CAPSIM/CAPSIM_V6//TOOLS/saxon.jar xprocess.s /c/CAPSIM/CAPSIM_V6//T
TOOLS/blockgen.xsl>xprocess.c
perl /c/CAPSIM/CAPSIM_V6//TOOLS/blockmaint.pl a xprocess.s
xprocess.s
BLOCK ALREADY EXISTS:xprocess
gcc -c -g -I/c/CAPSIM/CAPSIM_V6//include -I/c/CAPSIM/CAPSIM_V6//include -I../in
clude -I/c/CAPSIM/CAPSIM_V6//include/TCL/ -I/mingw/include xprocess.c
In file included from xprocess.c:31:
/c/CAPSIM/CAPSIM_V6/include/stars.h:37: warning: `IN' redefined
/mingw/include/rpcdce.h:9: warning: this is the location of the previous definit
ion
/c/CAPSIM/CAPSIM_V6/include/stars.h:38: warning: `OUT' redefined
/mingw/include/rpcdce.h:10: warning: this is the location of the previous defini
tion
ar -r libblock.a xprocess.o xsource.o zdummy.o

$ █
```

Figure 28 Make in BLOCKS Directory with *ZZZ* Removed from *xprocess.s* No Errors

Block Database Utility

A useful utility is the Block Database Management Utility. In this section we will only describe a few of its capabilities. Change directory to the BLOCKS directory and type the following command:

```
wish $CAPSIM/BLOCKS/blocksmanage.tcl
```

The Block Management Utility will appear as shown in Figure 29.

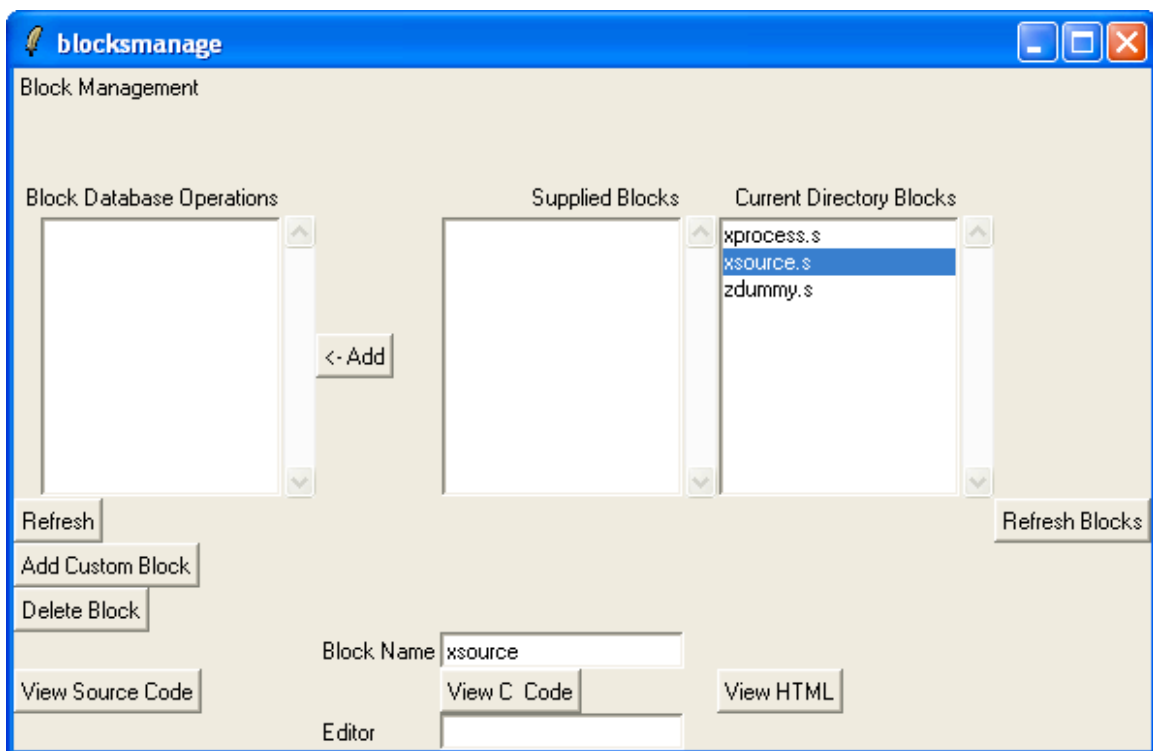


Figure 29 Block Management Utility

For now, only the blocks in the current BLOCKS directory are shown. The Supplied Blocks are blank. For more advanced operation this list will be populated with the supplied blocks. The three buttons:

- 1- View Source Code
- 2- View C Code
- 3- View HTML

are of interest. To view the source select the block and click on View Source Code. Note that the built in Windows editor *Notepad.exe* in the *c:\Windows* directory is used. You can change this by modifying the TCL source code for *blocksmanage.tcl*. For example you can use JEDIT or NOTETAB etc. Keep a backup of *blocksmanage.tcl* if you make this modification.

If you click on ViewHTML the HTML code for the selected block will be generated. For example, *xsource.htm*. To view it, open the BLOCKS directory from Windows Explorer and drop *xsource.htm* on FIREFOX or another Browser.

It is instructive to view the generated C code for the block. Note the STATES and PARAMETER data structures and the defines for parameters and states.