IDBUILD – Macro-Equation Processor

Inforum
September 2009

Idbuild is an adaptation of the Build program for aggregate model building to building Interindustry Dynamic models. Like Build, it translates the .sav files created by G into C++ code, builds a bank of all the scalar variables, and writes several files of C++ code for use in the simulation program. Also like Build, it requires a build.cfg file which specifies the name of the output (or workspace) bank and the initially assigned bank. Also like Build, it requires a Master file, but this file merely lists each .sav to be included in the preceded by the command "iadd". Unlike Build, the Master file should not contain identities or other code. Idbuild is invoked from G7 by the command Model | IdBuild. This command also proceeds to the compilation and linking of the model. (From DOS, IdBuild can be invoked by "idbuild master". The output files made by Idbuild are:

HIST.BNK A G standard bank containing all the series used or created by idbuild.
HIST.IND (As in Build, variables on the right of "fex" commands are not included.)

HEART.CPP A compilable C++ program. Each of the "iadd" files becomes a separate, callable function with a name derived from the name of the iadd file from which it was created. It also contains a function (tserin) to read in all the time series from bws and make them accessible everywhere in the forecasting program. Both past and preliminary future values are available at all times.

HEART.H Prototypes for all the functions in the heart.cpp file. This file makes it possible to call these functions from anywhere in the forecasting program.

TSERIES.INCA file to be included in the forecasting main module of the forecasting program declare the names of all the variables in HIST.BNK as variables which can be used in the program.

CALLALL.CPP A program to call all of the functions in HEART.CPP. It is used at the end of each years calculations to set or update rho adjustment factors.

The Master File for Idbuild

We will illustrate the forming of the master file with the Mudan model of China. The file for this model is:

```
bank cmdm
iadd invest.sav
iadd income.sav
iadd finance.sav
iadd pseudo.sav
```

```
q
```

Except for PSEUDO.SAV, the various .SAV files contain estimated equations. For example, invest.sav begins

```
title Other State-owned units investment
f sinvest = sibac$ + sirep$
r invn$35 =   -40.346479*intercept +
        0.110044*sinvest
```

The program recognizes PSEUDO.SAV as the name of a file with a special purpose. Namely, it puts into the model's G bank those time series which are not needed in any of the code-image equations but are required elsewhere, perhaps in identities or detached coefficient equations. For Mudan, the beginning and end of the pseudo.sav file are:

```
f trsa = trsa
f rpop = rpop
f upop = upop
...
f rscale = 1.
f uscale = 1.
```

These commands put the variables named trsa, rpop (rural population), and upop (urban population) into the model's G bank. At the end, it initializes the rscale and uscale variables to 1.0 in all years. Any right-hand side legal in G would be legal here.

Back in the master file, the final "q" signals the end of input to Idbuild. Idbuild will produce from this command the following heart.cpp file:

```
#include <stdio.h>
#include "constant.h"
#include "matrix.h"
#include "dyme.h"
#include "groups.h"
#include "databank.h"
#include "vamfile.h"
#include "fixbank.h"
#include "dyme.ext"
#include "heart.h"
#include "tseries.ext"
FILE *fmatrix;
int i,j,k,err;
extern int t;
float depend;
/* end of standard prolog */

void investf()
{
/*  Other State-owned units investment */
 sinvest[t]= sibac_[t]+ sirep_[t];
/* invn$35 */ depend =-40.346479+0.110044* sinvest[t];
      invn_35.modify(depend);
...
}

void incomef()
{
      ...
```

```
    }

void financef()
{
      ...
}
void tserin()
{
sibac_.in("sibac$");
sirep_.in("sirep$");
sinvest.in("sinvest");
...
}
```

Note that all of the "iadd" files have been turned into functions whose names are formed by adding an 'f' to the end of the "iadd" file name. (Any $'s in the file name have been turned to _'s; there are none in the example.) What were "f" commands have become C statements with the exception that the '$' character in variable names has been changed to an '_'. Examples of this change are seen in both the investf() function and the tserin() function. Any "fex" commands (none in the example) have disappeared, but the variable on the left of the "fex" has been created and entered into the data bank. Regression equations appear with the regression coefficients in the code. They calculate a variable, "depend", which is passed to the routine modify(), along with the identification number of the dependent variable. The modify() routine then looks to see if there are any macro variable fixes -- add, multiply, index, growth, skip, or rho adjustment -- on that variable, and then stores the variable with the appropriate modification, if any. At the end of the HEART.CPP file is the tserin() function. It is called at the beginning of a run of the model to read into memory all the time-series variables. Finally, note that it contains all the variables which are in the PSEUDO.SAV file, even though no function was created by this file. This is the way to put into the data bank those variables, such as labfor, which appear in no code-image equation. The name "pseudo" is a keyword to the program; files by any other name create functions.

The HEART.H file, created by idbuild for this example, is:

```
void investf();
void incomef();
void financef();
void exdgf();
```

It simply provides the prototypes required by C++ in any program which uses the functions in the HEART.CPP file.

The TSERIES.INC file is

```
Tseries sibac_, sirep_, sinvest, invn_35, invn_36, d88, d90, invn_37,
rni, rpindex, rpop, rincome_, trsa, trsa_, invn_38, ulfi,
...
uscale;
```

These files are "#included" in the forecasting program to declare that sibac_, sirep_, etc. are objects of the form "Tseries". A "Tseries" is an "object" defined in the forecasting program

designed to hold a time series. One of the things that a Tseries object "knows" how to do is to load itself from the assigned G data bank. Thus, in the tserin() function in the heart.cpp program above, the command

```
ngdpc.in("ngdpc");
```

tells the ngdpc object to read in its data contents from the series called "ngdpc" in the data bank.

The final product of Idbuild is the CALLALL.CPP file, which, for our example, is simply

```
#include "heart.h"
void callall()
{
investf();
incomef();
financef();
exdgf();
}
```

As its name suggests, callall() is simply a program to call all of the functions in the heart.cpp file. Its function is in connection with rho adjustments, as will be seen in the forecasting program.

Combining Vector and Tseries Variables in a Function With Idbuild

When building interindustry macro models one usually needs to integrate the macro and the industry computations. For example, it is often necessary to form a macrovariable as a sum of components of a vector. Conversely, it may be that some sectoral variable is required on the right hand side of a macro equation. When this is the case, the Idbuild command “isvector” is particularly useful. This command indicates to Idbuild that a variable in one of the following save files is to be treated as an element of a vector, and not as a macrovariable, or Tseries variable, which is the default. For example, in the LIFT model of the U.S., the equation for railroad construction uses output of sector 59 (Railroads). Other equations use aggregates of output of many sectors. The included sections of files below show how this is handled. In G, here is part of the regression file, CONSTR.REG:

```
save constr.sav
f outman = @csum(out,9-58)/1000.
f outbus = @csum(out,64,65,72,73,77-80)/1000.
f outtrade = @csum(out,69-71)/1000.
f outmin = @csum(out,2-6)/1000
#==============================
ti 15. Railroad Construction
r cst15$ = cstoth, rpoil[2], rcbr[1], out59, doutrail, doutrail[1]
gr *
```

Here is a small MASTER file, which will generate the code for this function only.

```
ba constr
isvector out,emp,pdm,cstk
```

```
iadd constr.sav
```

Here is the code for the constrf() function. Note that since the "isvector" command was in effect, Idbuild knows that "out" is a vector. Therefore, it passes "out" as one of the vectors in the argument list to constrf(), it writes out the csum function correctly as a method of type Vector, and it writes "out[59]" on the right hand side of the regression equation instead of "out59[t]".

```
void constrf(Vector& out,Vector& emp,Vector& pdm,Vector& cstk)
{
 outman[t]=out.csum("9-58")/1000.;
 outbus[t]=out.csum("64,65,72,73,77-80")/1000.;
 outtrade[t]=out.csum("69-71")/1000.;
 outmin[t]=out.csum("2-6")/1000;
        .
        .
        .
/*  15. Railroad Construction */
/* cst15_ */ depend =3699.702916+-0.405094* cstoth[t]+5.812924* rpoil[t-2]+
17.332162* rcbr[t-1]+0.020963* out[59]+0.040612* doutrail[t]+
0.117446* doutrail[t-1];
        cst15_.modify(depend);
        .
        .
        .
}
```

At the present time (Interdyme version 2.2), Idbuild doesn't know how to handle lagged values of vector variables. In this case, you can make the vector variable a macrovariable, and include code in your model to copy the macrovariable values to the vector, and vice versa. For example, another regression in the construction equations mentioned above uses construction capital stock of category 16 lagged once ("cstk16[1]"). The way to handle this is as follows. Before opening up the save file in G, first do:

```
f cstk16$ = cstk16
```

Then, include cstk16$[1] on the right hand side of the equation. In the simulation model, remember to fill up the macrovariable with the value of construction capital stock of category 16 before calling the construction function:

```
cstk16_[t] = cstk[16];
constrf(out,emp,pdm,cstk);
```

Note that in addition to the "iadd" command, IdBuild has one more command not found in standard Build. That is the "break" command. Its format is:

break <filename>

After the "break"command, subsequent C++ code will go to the named file, with the extension .CPP appended, rather than to HEART.CPP. The reason for this command is that with a large model, HEART.CPP can become too large, and may fail to compile. Also, it is somtimes more convenient to group the functions written by IdBuild into smaller logically organized files.

Macrovariables can also be used on the right hand side of equations for vector variables, called “detached-coefficient equations”. These are described in the next section.

Please visit the Software pages of the Inforum web site for more details and to download the Fixer software: www.inforum.umd.edu/software/software.html.