

LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
-LIGO-
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Technical Note	LIGO-T040005- v2	07/21/09
Runtime Dynamic Linking in E2E - all about FUNC_X -		
Melody Araya and Hiro Yamamoto Caltech		

This is an internal working note
of the LIGO Project.

California Institute of Technology
LIGO Project - MS 51-33
Pasadena CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project - MS 20B-145
Cambridge, MA 01239
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

WWW: <http://www.ligo.caltech.edu>

ABSTRACT

The End to End (E2E) code has added a new primitive module group which allows C++ syntax in its function modules. The user provides C++ code 'snippets' in the primitive's settings and, without any user intervention, C++ source code is dynamically generated, compiled, linked, and accessed during runtime.

The new primitive group is called FUNC_X which includes FUNC_X_1x1, FUNC_X_2x2, FUNC_X_4x4, FUNC_X_8x8, FUNC_X_16x16, FUNC_X_VxV, and FUNC_X_2VxV.

This document first gives simple examples how to use the new E2E primitives. Then it describes how the E2E code uses runtime dynamic linking with FUNC_X to provide a more efficient and more flexible function modules for the simulation.

INTRODUCTION

Generic function modules have been provided in the E2E software where the mathematical calculations were written by the user in a syntax similar to the C language. The LIGO--T00047-01 document (e2e primitive module Reference Manual) describes the Generic function modules, FUNC_xxx. The new FUNC_X improves on the concept by providing more flexibility to the user and a more efficient processing.

Motivation

Flexibility - The previous FUNC_xxx implementation allowed for a syntax similar to the C language. The user was limited to the list of allowable constructs when writing the function module. The new FUNC_X uses the C++ compiler to produce the runtime dynamic library. Therefore, any C++ language constructs are allowed in the primitive's settings including all classes already defined in the E2E code such as *field* and *Matrix*. The user only needs to write code 'snippets' or the body of a C++ function.

Efficiency - In the previous FUNC_xxx implementation, the setting, Equation, was parsed and evaluated in each occurrence of the primitive. The new FUNC_X converts the settings into the native machine code. This machine code is dynamically linked with the E2E application and is executed when the primitive is used. Any optimizations are also applied to the primitive's settings.

In this document

Getting Started

A Simple Example

A More Advanced Example - How to create a digital filter

FUNC_X Primitive Settings

FUNC_X Primitive Settings Window

Steps to use FUNC_X

Build E2E - what are the changes introduced by having a dynamic library

Applications using the dynamic library - for now just *modeler*

Modeler - changes seen by the user

FUNC_X Architecture

Dynamic Library Implementation

Generated Code

Troubleshooting

GETTING STARTED

The FUNC_X code relies on a new environment variable called **E2E_HOME**. This directory path contains the E2E header files, library archive, primitives, and executables. The README file in the E2E package has a step-by-step explanation on how to define this when the package is installed.

Various settings are used in the following way in a .h and .cc file, where "==> xxx" means xxx is inserted in that location. C++ source codes are created in e2eBoxes in the directory where the main box exists. @@CLASS is the class name assigned to each FUNC_X using a hash algorithm. When you need to use a class name, use @@CLASS for it.

File @@CLASS.h

==> **Header**

class @@CLASS

```
{  
    void action( ... );  
    ==> MemberDecl  
    ==> Global  
}
```

File @@CLASS.cc

```

#include "@@CLASS.h"
@@CLASS::action( ... ) : called at each time step
{
    ==> Equation
}
@@CLASS::@@CLASS( void )
{
    ==> Constructor : called only once before anything else
}
@@CLASS::~~@@CLASS( void )
{
    ==> Destructor
}
==> MemberImpl

```

SIMPLE EXAMPLES

To use FUNC_X, mathematical calculations are defined in the Equations setting, much like the previous FUNC_XXX implementation. The table below shows some differences between two versions.

	<i>Old Style</i> (FUNC_1x1, etc)	<i>New Style</i> (FUNC_X_1x1, etc)	<i>Comments</i>
Simple math	Fill in the Equations setting to <code>out0 = 1/sqrt(1 + pow(in0,2));</code>	Fill in the Equations setting to <code>out0 = 1/sqrt(1 + pow(in0,2));</code>	
Timer	Fill in the Equations setting to <code>startTime = 0.1;</code> <code>out0 = if (time_now() > startTime, in0, 0);</code>	Fill in the Equations setting to <code>/* this is C++ code so, all variables need to be declared with the type before being used */</code> <code>double startTime = 0.1;</code> <code>/* this is C++ (condition ? yes_case: no_case syntax) */</code> <code>out0 = time_now > startTime ? in0 : 0;</code>	time_now is the current time. See the full description given later to see how it is passed.
Macro	Fill in the Equations setting to <code>out0 = in0 * sin(2 * PI * freq * time_now());</code>	Set the Equations setting to <code>out0 = in0 * sin(2 * M_PI * freq * time_now);</code>	freq is a macro value defined in <i>e2eDB.mcr</i> . This needs to be declared in Global setting. M_PI is a constant declared in a g++ header file

	<i>Old Style</i> (FUNC_1x1, etc)	<i>New Style</i> (FUNC_X_1x1, etc)	<i>Comments</i>
		<p>Set the Global setting to</p> <pre> /* To access a macro value, delared it here. This setting is only to access macro values. To declare variables to be used among member functions in this FUNC_X, declare it in MemberDecl setting. */ adlib_real freq; </pre>	

ADVANCED EXAMPLE

The following example is an example of creating a digital filter in the new FUNC_X implementation.

Digital Filter Old Style	Comment
<pre> Fill in the Equations setting to seismicLow(x) = digital_filter(0.9 * 2e-7*PI, {}, {-0.2*PI}, {}, {(-0.126,0.942)}); seismicHigh(x) = digital_filter(4e-6, {}, {-20*PI,-20*PI}); out0 = seismicLow(white_noise(in0)) + seismicHigh(white_noise(in0)); </pre>	

Digital Filter New Style	Comments
<pre> Fill in the Header setting to // e2eRand class is defined in this header #include "random.h" Fill in the MemberDecl setting to /* member data and function declaration */ // digital filter objects e2e_dfKernel seismicLow, seismicHigh; DF seismicLow, seismicHigh; // white_noise function adlib_real white_noise(adlib_real amp); </pre>	<p>Object declarations</p> <p>seismicLow and seismicHigh are e2e_dfKernel objects.</p> <p>Member function white_noise</p> <p>@@CLASS is used to fill in the class name.</p>

Digital Filter New Style	Comments
<p>Fill in the MemberImpl setting to</p> <pre> /* 1) Implementation of white_noise member function. 2) @@CLASS is used to represent a class of this FUNC_X 3) If a static function is to be implemented, implement it without a class name. 4) TIME_STEP is a special constant which is the time step for this module to be ticked. 5) 1 / sqrt(TIME_STEP) is needed to generate properly normalized density distribution, N/second. */ adlib_real @@CLASS::white_noise(adlib_real amp) return amp*2*rand_uniform() / sqrt(TIME_STEP); </pre>	<p>TIME_STEP is a special macro which does NOT need to be declared in the Global setting.</p>
<p>Fill in the Constructor setting to</p> <pre> /* Any one time initializations need to be placed here. */ /* The setting for low frequency component */ seismicLow.setGain(0.9 * 2e-7*M_PI); // digital filter gain seismicLow.addPole(-0.2*M_PI); // add a pole at -0.2*M_PI /* Add a complex pole pair. This add poles at -0.126+i*0.942 and at -0.126- i*0.942 */ seismicLow.addPolePair(adlib_complex(-0.126, 0.942)); /* This initialize internal data using the digital time step of TIME_STEP */ seismicLow.calcAB(TIME_STEP); /* The setting for high frequency component */ seismicHigh.setGain(4e-6); /* One can add multiple poles and zeros by calling addXXX function repeatedly. */ seismicHigh.addPole(-20*M_PI); seismicHigh.addPole(-20*M_PI); // Don't forget to initialize seismicHigh.calcAB(TIME_STEP); </pre>	<p>Any one time initialization needs to be placed in this section</p> <p>In this case seismicLow and seismicHigh initialization happens here.</p> <p>The class e2e_dfKernel has a method called filterApply</p>
<p>Fill in the Equations setting to</p> <pre> // Apply the filters out0 = seismicLow.filterApply(white_noise(in0)) + seismicHigh.filterApply(white_noise(in0)); </pre>	

FUNC_X PRIMITIVE SETTINGS

Equations

FUNC_X Primitive Settings

(Mandatory) Mathematical calculations needed to be made by the primitive module.

Scalar input is designated by `in0`, `in1`, ...

Scalar output is designated by `out0`, `out1`, ...

Vector input is designated by `inVec0` and `inVec1`

Vector output is designated by `outVec0`

The variable `time_now` can be used to retrieve the current time in the time domain loop. The user does not need to declare `time_now` since the `FUNC_X` method passes it as an argument.

Global

Global variable declarations used in the macro definitions.

These are inserted in the C++ header file as private data members for the dynamically created class. An example of the setting would be:

```
double HZ;  
double SeismicNoiseAmp;  
double ThermalNoiseAmp;  
double FreqNoiseAmp;
```

The variable names should be identical to the macro names. For the example above, there exists macro values for `HZ`, `SeismicNoiseAmp`, `ThermalNoiseAmp`, and `FreqNoiseAmp` in a macro database that had been included. As a rule of thumb, when the box file contains a macros settings, the variables need to be declared in this section. Also if any macros need to be accessed in any of the Settings (Equations, MemberImpl, Constructor, or Destructor), the declarations should be included in this section.

The macro values are retrieved in the class' constructor.

TIME_STEP is a special macro which does not need to be declared. It is the time step for the module to be ticked.

MemberDecl

C++ header file declarations. This would include variables and member function declarations. Variables declared in this section are generally used in one or more sections of the class.

MemberImpl

C++ source file implementation. The source code of the member function implementation from the functions declared in **MemberDecl**.. This portion of the Settings is going to be 'copied' to a .cc file. Since C++ syntax dictates that class name prepend the function member name, the user will have to use temporary class name "`@@CLASS`". An example of a member implementation is:

```
double @@CLASS::length(double x, double y)
```

```

{
    return sqrt( x*x + y*y);
}

```

Constructor

Class constructor definition. This setting is used for any one-time initialization steps. The user can assured that the macro database has been read in before these initialization steps are performed.

**Destructor
Header**

Any cleanup which are required for the class.

Additional header files which are needed by the class. This header file definition will be added to the generated header code's list of #include. The Header Setting should be defined as it should appear in the header file. An example of the setting would be:

```

#include <list>
#include "random.h"

```

Several header files have already been included in the generated files. The user does not need to include these files.

```

#include <string>
#include <vector>
#include <iostream>
#include "e2eDB.h"
#include "e2e_dfKernel.h"
#include "adlib_types.h"

```

} already included in the header files

```

#include <time.h>
#include <math.h>

```

} included in source files

DoWhenGlobalChanges

Mathematical operations which need be calculated whenever the macro values change. This code is inserted in the setGlobals method after all the macro values have been retrieved.

showInstanceSettings

A flag to enable the simplified FUNC_X display in the Primitive Setting Window

FUNC_X PRIMITIVE SETTINGS WINDOW

Just like any E2E primitives, the FUNC_X primitives are modified using the ALFI graphical interface. However, unlike other primitives, FUNC_X has two types of graphical user interface to choose from. One display shows all the primitive settings variables discussed in the previous pages. The other display a simplifies the FUNC_X display. It allows the user to modify the variables used in the MemberDecl setting and hides all other primitive settings. Therefore, instead of seeing the values of Equations, MemberImpl, MemberDecl, etc in the FUNC_X primitive settings window, only the variables used in the MemberDecl setting are displayed.

The simplified FUNC_X display is enabled when the primitive setting `showInstanceSettings` is set to `true`.

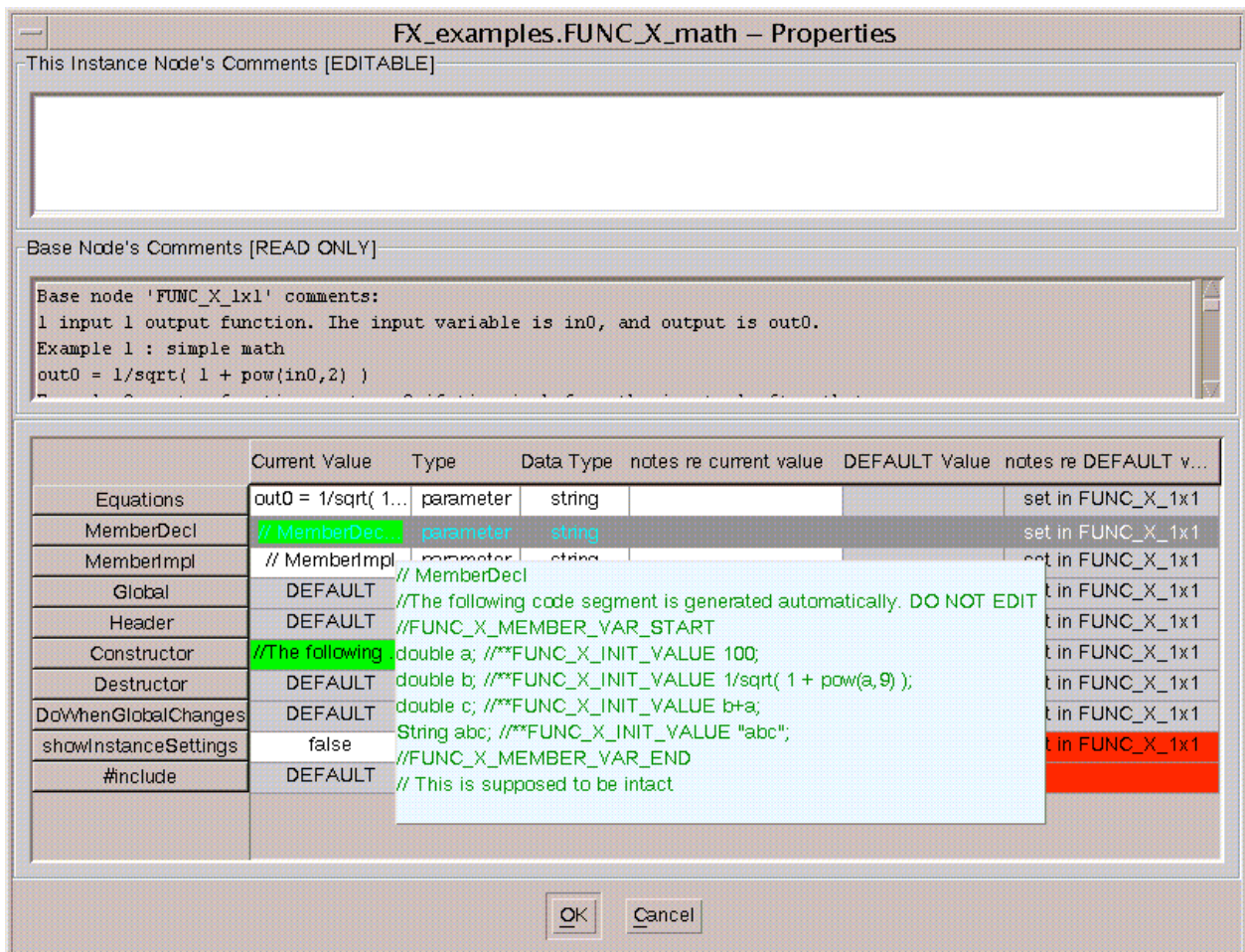
To specify the variables which can be displayed in the simplified format, place the variables in the `MemberDecl` setting enclosed inside the following markers.

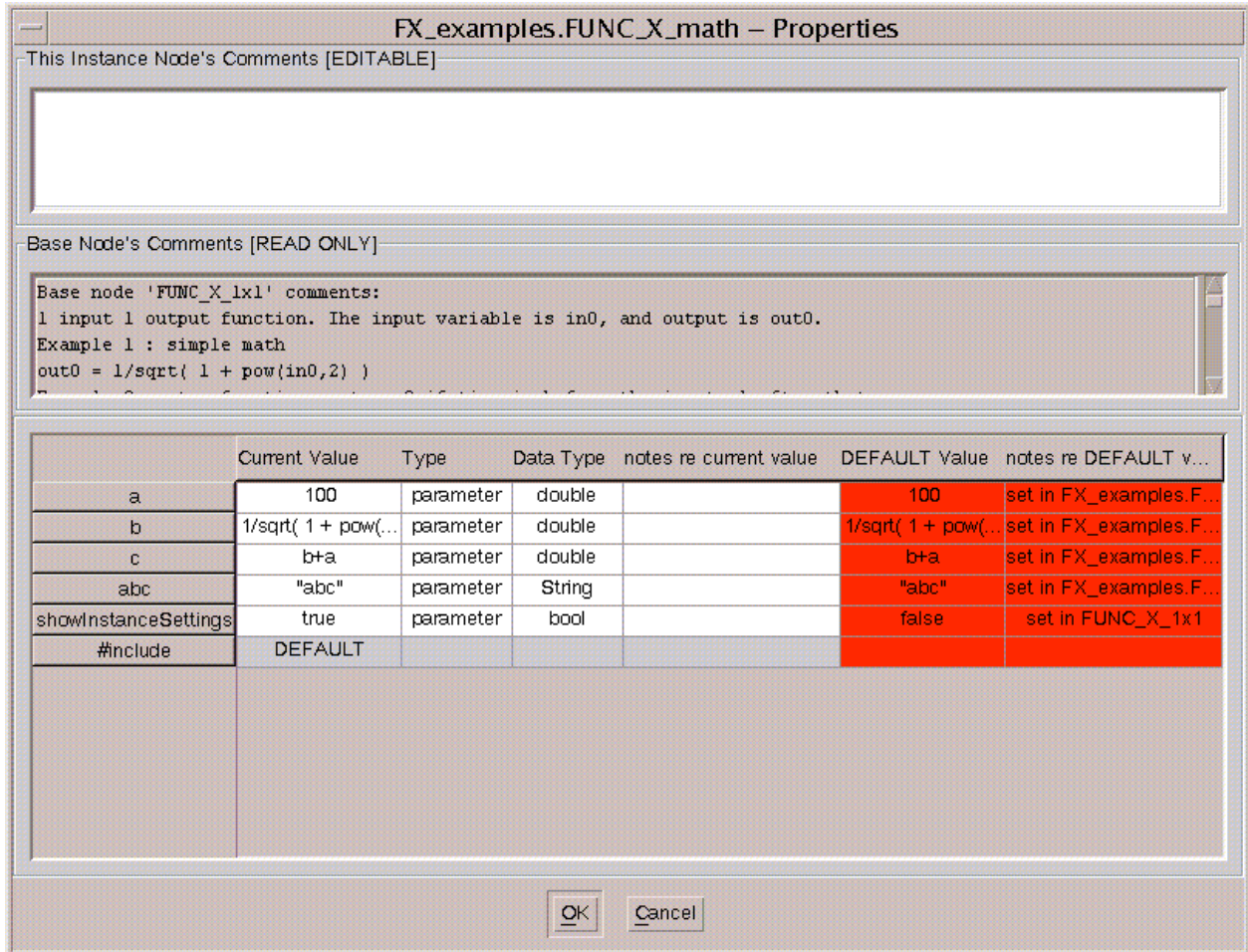
```
//FUNC_X_MEMBER_VAR_START
//FUNC_X_MEMBER_VAR_END
```

To initialize the values, place the values after `/**FUNC_X_INIT_VALUE`. Don't forget the semicolon at the end of the value.

Example setting for `MemberDecl`:

```
//FUNC_X_MEMBER_VAR_START
double a;      /**FUNC_X_INIT_VALUE 100;
double b;      /**FUNC_X_INIT_VALUE 1/sqrt( 1 + pow(a,9) );
double c;      /**FUNC_X_INIT_VALUE b+a;
String abc;    /**FUNC_X_INIT_VALUE "abc";
//FUNC_X_MEMBER_VAR_END
```





The Primitive Setting window when `showInstanceSettings` is set to true.

BUILD PROCESS

A new environment variable needs to be defined for the FUNC_X code:

- E2E_HOME** - this should reference the base directory the location of the library, executables, and include files.

In order to compile and link the C++ code generated from the *modeler*, the E2E software need to figure out what the compiler's name and location, compiler and link options, library archive locations, and include files' location. The E2E_HOME environment variable allows the E2E software to find out these crucial information.

When the user configures the build environment, either by the *configure* or *build-e2e* scripts, the compiler and linker settings are written out to the files *cc-settings* and *lib-settings*, respectively. The E2E software reads these files to build the dynamic library.

Build Summary

```
> setenv E2E_HOME /home/user/my_e2e
> cd $E2E_HOME
> ./build-e2e -install
```

The *install* process creates three (3) directories in the E2E_HOME directory, *include*, *lib*, and *bin*.

`$E2E_HOME/bin` - contains the executables (*modeler*, *modeler_freq*, etc).

`$E2E_HOME/lib` - contains *libAdlibMM.a*, *cc-settings*, *lib-settings*. As well as the *prm* files and *xbm* files.

`$E2E_HOME/include` - contains the include files which correspond to *libAdlibMM.a*

A simple way to make sure the environment is properly set is to include the following lines in your `.cshrc` file:

```
setenv E2E_HOME /home/user/my_e2e
setenv PATH ${E2E_HOME}/bin:${PATH}
setenv E2E_PATH .:${E2E_HOME}/lib
```

MODELER

Currently the only E2E application which use the dynamic library is the *modeler*. From a user's perspective, no additional setup is introduced by the *modeler*. It requires no user intervention when it comes to the creation and use of the dynamic library. The software creates the required directories and files and initializes them before the time domain loop starts.

The *modeler* creates the dynamic library the first time any FUNC_X primitives are used. It prints out "Creating the Shared Library...." when building the dynamic library. And "Done Creating the Shared Library...." when it has been successfully built.

If any errors are encountered, the C++ error message is displayed and the *modeler* exits. The user will need to review the FUNC_X primitive settings and fix any syntax errors.

As long as the FUNC_X settings are unchanged, the *modeler* merely opens the shared object without creating any source files. When a FUNC_X definition changes, the *modeler* updates only the modified source files, cleans up any outdated source files, compiles the new source, and recreates the shared object automatically.

FUNC_X ARCHITECTURE

Dynamic Library Implementation

The E2E code makes use of the C++ *libdl* interface. It uses *dlopen()* to load a dynamic shared object (DSO), *dlsym()* to find a symbol in loaded DSO, *dlclose()* to unload, and *dLError()* to report errors. The *configure* script used in E2E checks for the existence of the *libdl* interface. The following message should be displayed when *configure* is invoked.

```
Checking for main in -lrt... yes
```

```
checking for main in -ldl... yes
```

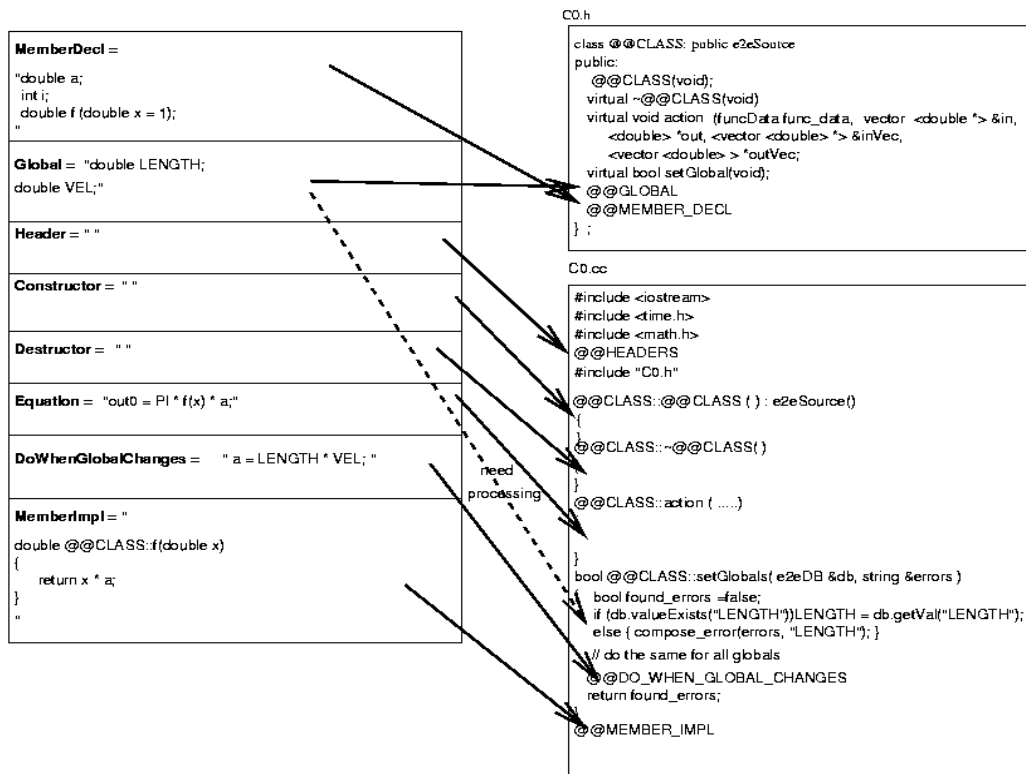
```
checking for main in -liberty... yes
```

```
checking for main in -lbfd... yes
```

```
checking for main in -lcwd... no
```

The software needs the *libdl* interface in order to execute the FUNC_X dynamically created library.

The diagram below shows the the code snippets from the primitive settings are positioned in the C++ source code which are dynamically created.



MODELER BEHIND THE SCENES

When FUNC_X primitives are used in the *modeler*:

- the *modeler* creates a subdirectory named `e2eBoxes` from the current working directory (usually where the box files are located). The `e2eBoxes` directory will contain the FUNC_X C++ source code.
- After the box files are all read into the system, the C++ source files are compiled to create a shared object named `libe2e.so`. If any errors occur during the compile process, the C++ compiler message is displayed and the application halts.
- Right before entering in the time domain loop, the FUNC_X objects from the shared library are instantiated.
- During the instantiation process, the constructors for each of the FUNC_X retrieves the macro values from the macro database to initialize the variables in Globals setting.
- Each FUNC_X class has a public method called `setGlobals` to explicitly read the macro values. Its prototype is: `bool setGlobals (e2eParamDB *db, string *error_message);` The method returns *true* if the values have been successfully retrieved. If the function returns a *false* the `error_message` string contains some information in determining the problem.

- When the *modeler* finishes its time domain loop, the FUNC_X's Destructor is called and the dynamic library is closed thereafter.

TECHNICAL TROUBLESHOOTING

1. When pages and pages of error messages are seen during the shared object is being created, make sure that the environment variable E2E_HOME is set properly and that the `${E2E_HOME}/include` contains the E2E header files and that `${E2E_HOME}/lib` contains the `libAdlibMM.a`, `cc-settings`, and `lib-settings` files.

Addition – needs to be included in the main document

Using separate files, x.cc and x.h to implement C++ codes to make the code easier to read

Generic explanation and templates

`#{E2E_HOME}/lib/ FUNC_X_comment.txt`

--- How various settings are used in C++ codes ---

`@@INCLUDE filename`

will insert the content of the file without using preprocessor.

In files included this way, `@@CLASS` can be used to specify the class name which is automatically created by the program.

`/*@@BEGIN_COMMENT*/` and `/*@@END_COMMENT*/` are replaced by `/*` and `*/` respectively. The equation code can be stored in a cc file in the following format, and the editor will recognize it as a legitimate cc function, while only MAIN CODE are used as the equation code. Green sections below become comments and only the red section is treated as necessary codes.

```
/*@@BEGIN_COMMENT*/
void @@CLASS::action
( vector <double *> &in, vector <double> *out,
  vector <vector <double > *> &inVec, vector <vector <double> > *outVec )
{
/*@@END_COMMENT*/
```

MAIN CODE

```
/*@@BEGIN_COMMENT*/
}
/*@@END_COMMENT*/
```

In the .h and .cc files, sections marked by `==>` are to be filled.

--- `@@CLASS.h` : created in e2eBoxes/ -----

`==> Header`

```
class @@CLASS
{
  void action( ... );
  ==> MemberDecl
```

--- To specify which variables are going to be displayed in a simplified form, enclose them in the following markers

```
//FUNC_X_MEMBER_VAR_START
//FUNC_X_MEMBER_VAR_END
```

--- To initialize the values, place the values after **/**FUNC_X_INIT_VALUE**
--- Don't forget the semicolon at the end

Example:

```
//FUNC_X_MEMBER_VAR_START  
double a;    /**FUNC_X_INIT_VALUE 100;  
double b;    /**FUNC_X_INIT_VALUE 1/sqrt( 1 + pow(a,9) );  
double c;    /**FUNC_X_INIT_VALUE b+a;  
String abc;  /**FUNC_X_INIT_VALUE "abc";  
//FUNC_X_MEMBER_VAR_END
```

==> Global

```
}
```

--- @@CLASS.cc : created in e2eBoxes/ -----

```
#include "@@CLASS.h"  
@@CLASS::action( ... ) : called at each time step  
{  
    ==> Equation  
}  
@@CLASS::@@CLASS( void )  
{  
    ==> Constructore  
}  
@@CLASS::~~@@CLASS( void )  
{  
    ==> Destructor  
}  
==> MemberImpl
```

--- To display the settings in a simplified form, set showInstanceSettings to true

==> showInstanceSettings = true

[\\${E2E_HOME}/lib/ FUNC_X_SettingTemplate.h](#)

```
/* ===== */  
/* ===== MISC HEADER COMPONENT ===== */  
/* ===== */  
  
/*@@BEGIN_FX_HEADERS*/ /* ++++++ */  
  
/* insert your miscellaneous header par here */  
  
/* ----- */ /*@@END_FX_HEADERS*/
```

```
using namespace std;  
class @@CLASS: public e2eSource {
```

```

public:
const adlib_real TIME_STEP;

/* ===== */
/* ===== GLOBAL ===== */
/* ===== */

/*@@BEGIN_FX_GLOBAL*/ /* ++++++ */

/* insert your globals for accessin macro definitions here */

/* ----- */ /*@@END_FX_GLOBAL*/

/* ===== */
/* ===== MEMBER DECLARATION ===== */
/* ===== */

/*@@BEGIN_FX_MEMBER_DECL*/ /* ++++++ */

/* insert your member declaration here */

/* ----- */ /*@@END_FX_MEMBER_DECL*/

};

```

[\\${E2E_HOME}/lib/ FUNC_X_SettingTemplate.cc](#)

```

#include "@@CLASS.h"

/* ===== */
/* ===== CONSTRUCTOR ===== */
/* ===== */

void @@CLASS::@@CLASS( FUNC_X_data_type *fxdata_arg )
{
/*@@BEGIN_FX_CONSTRUCTOR*/ /* ++++++ */

/* insert your constructor here */

/* ----- */ /*@@END_FX_CONSTRUCTOR*/
}

/* ===== */
/* ===== DESTRUCTOR ===== */
/* ===== */

void @@CLASS::~@@CLASS( void )
{

```

```

/*@@BEGIN_FX_DESTRUCTOR*/ /* ++++++ */

/* insert your destcutor code here */

/* ----- */ /*@@END_FX_DESTRUCTOR*/
}

/* ===== */
/* ===== CODE TO HANDLE GLOBAL CHANGHE ===== */
/* ===== */

bool @@CLASS::setGlobals( e2eParamDB *db, string *error_message )
{
    bool no_errors = true;
    @@GET_DB_VALS;

/*@@BEGIN_FX_DO_WHEN_GLOBAL_CHANGES*/ /* ++++++ */

/* insert your code when macro setting is changed here */

/* ----- */ /*@@END_FX_DO_WHEN_GLOBAL_CHANGES*/

    return no_errors;
}

/* ===== */
/* ===== MAIN CODE ===== */
/* ===== */

void @@CLASS::action
( vector <double *> &in, vector <double> *out,
  vector <vector <double > *> &inVec, vector <vector <double> > *outVec )
{
/*@@BEGIN_FX_EQUATION*/ /* ++++++ */

/* insert your main action code here */

/* ----- */ /*@@END_FX_EQUATION*/
}

/* ===== */
/* ===== PREACTION CODE ===== */
/* ===== */

void @@CLASS::preaction (vector<data_ref>* outputs);
{
/*@@BEGIN_FX_PREACTION*/ /* ++++++ */

```

```

/* insert your preaction code here */

/* ----- */ /*@@END_FX_PREACTION*/
}

/* ===== */
/* ===== MEMBER FUNCTION IMPLEMENTATION === */
/* ===== */

// C++ code of member function implementation
// void @@CLASS::func1( .... ) {}

/*@@BEGIN_FX_MEMBER_IMPL*/ /* ++++++ */

/* insert your member implementation code here */

/* ----- */ /*@@END_FX_MEMBER_IMPL*/

```

Easier explanation

File @@CLASS.h

==> **Header**

```

class @@CLASS
{
    void action( ... );
    ==> MemberDecl
    ==> Global
}

```

File @@CLASS.cc

#include "@@CLASS.h"

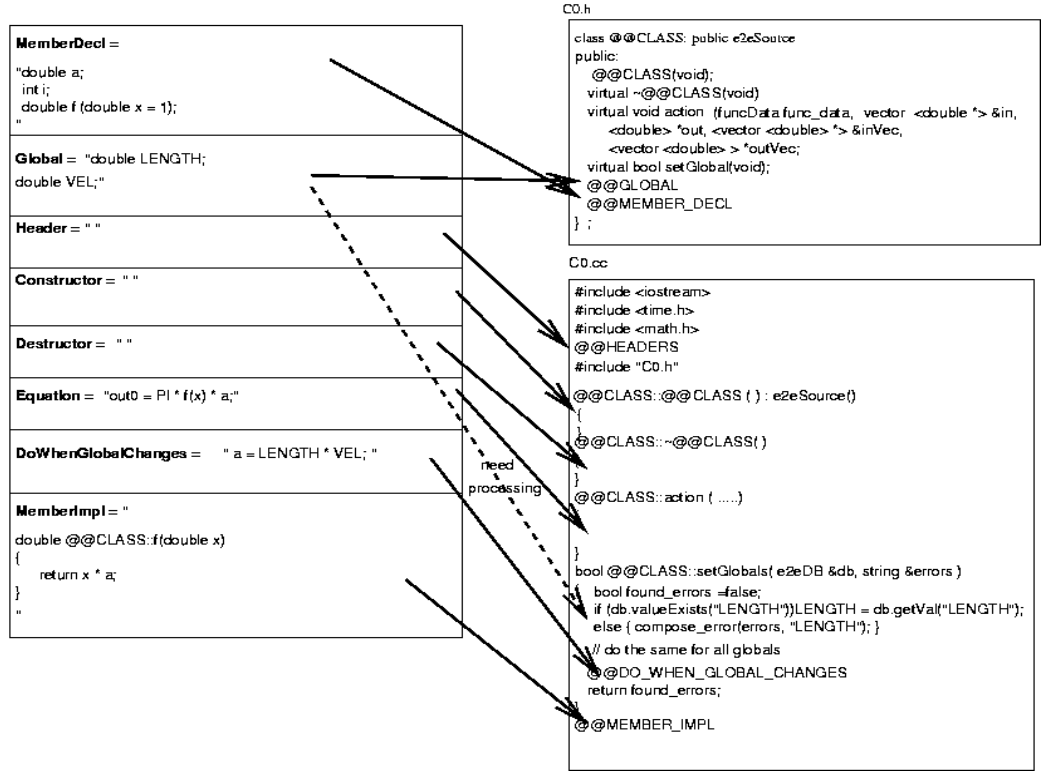
@@CLASS::action(...) : called at each time step

```

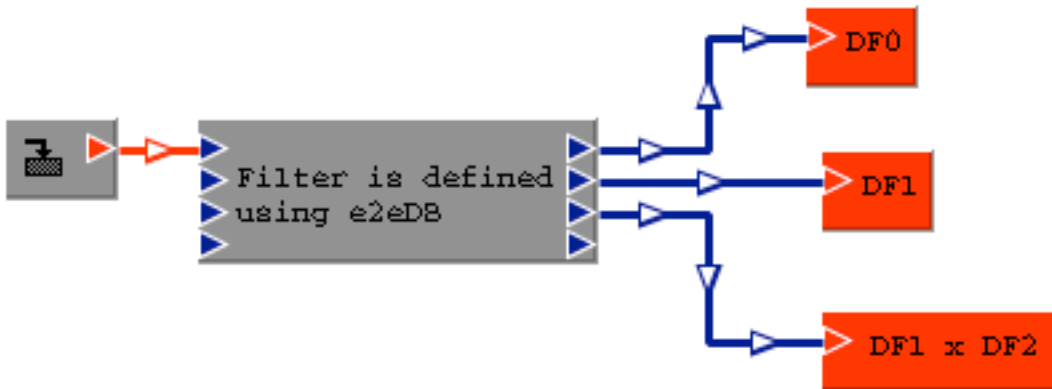
{
    ==> Equation
}
@@CLASS::@@CLASS( void )
{
    ==> Constructor : called only once before anything else
}
@@CLASS::~@@CLASS( void )
{
    ==> Destructor
}

```

}
 ==> MemberImpl



A FUNC_X example using digital filters. The filter is defined in e2eDB.mcr using e2e macro.



	Current Value	Type
Equations	@@INCLUDE CC_X_main.cc	par
MemberDecl	@@INCLUDE CC_X_memberDecl.h	par
MemberImpl	DEFAULT	par
Global	@@INCLUDE CC_X_global.h	par
Header	DEFAULT	par
Constructor	@@INCLUDE CC_X_constructor.cc	par
Destructor	DEFAULT	par
DoWhenGlobalChanges	DEFAULT	par
showInstanceSettings	DEFAULT	par
#include	DEFAULT	par

@@INCLUDE filename includes a file and does necessary substitution, including @@CLASS.

CC_X example

FUNC_X replaces `/*@@BEGIN_COMMENT*/` by `/*` and `/*@@END_COMMENT*/` by `*/`.

The content of the following CC_X_main.cc is a legitimate cc file, and a language sensitive editors, like emacs, handles it appropriately. In FUNC_X, only the blue part is used as a code.

CC_X_main.cc : main action called at each time step

```
/*@@BEGIN_COMMENT*/
void @@CLASS::action
( vector <double *> &in, vector <double> *out,
vector <vector <double > *> &inVec, vector <vector <double> >
*outVec )
{
/*@@END_COMMENT*/

    double noise = WhiteNoiseFunc( in0 ); // member function
                                           // declared in memDecl

    // filter related functions are listed in e2eREADME.txt
    out0 = DF0.filterApply( noise );      // filters defined in
    out1 = DF1.filterApply( noise );      // constructor
    out2 = DF0Copy.filterApply( out1 );   // using macro values

/*@@BEGIN_COMMENT*/
}
/*@@END_COMMENT*/
```

CC_X_global.h : e2e macro names need to be declared here

```
// e2e macros are all reals.
double NumZeros0;
double NumZeros1;
double NumPoles0;
double NumPoles1;

// macro names are listed one per line
double gain0;
double zero00;
double zero01;
double pole00;
```

```
double pole01;  
double zQ00;  
double zQ01;  
double pQ00;  
double pQ01;
```

```
double gain1;  
double zero10;  
double zero11;  
double pole10;  
double pole11;  
double zQ10;  
double zQ11;  
double pQ10;  
double pQ11;
```

**e2eDB.mcr : all macro names declated in global section
need to be defined as macro names.**

```
NumZeros0 = 2  
NumZeros1 = 1  
NumPoles0 = 2  
NumPoles1 = 2
```

```
% macro names are listed one per line
```

```
gain0 = 1000  
zero00 = 0  
zero01 = 10  
pole00 = 100  
pole01 = 100  
zQ00 = 0  
zQ01 = 0  
pQ00 = 0  
pQ01 = 0
```

```
gain1 = 1000  
zero10 = 10  
zero11 = 20  
pole10 = 100  
pole11 = 1000  
zQ10 = 0  
zQ11 = 0  
pQ10 = 0  
pQ11 = 0
```

CC_X_memberDecl.h : member data and member functions declared (and defined as inline functions)

```
// digital filters
DF DF0, DF1, DF0Copy;

// member functions
adlib_real WhiteNoiseFunc( adlib_real amp )
{
    return amp*e2eRand::rndnorm( ) / sqrt( TIME_STEP );
}

// build digital filter up to 2 zeros and poles
void buildDF
( DF* dfp, string name,
  double gain, int NumZeros, int NumPoles,
  double z0, double z1, double zQ0, double zQ1,
  double p0, double p1, double pQ0, double pQ1 )
{
    double zs[2], zQ[2], ps[2], pQ[2];

    zs[0] = z0; zs[1] = z1; zQ[0] = zQ0; zQ[1] = zQ1;
    ps[0] = p0; ps[1] = p1; pQ[0] = pQ0; pQ[1] = pQ1;

    dfp->addKZPHzQ( gain, NumZeros, zs, zQ, NumPoles, ps, pQ );
    dfp->setName( name );
}
}
```

CC_X_constructor.cc : this is called once before action starts

```
/*@@BEGIN_COMMENT*/
void @@CLASS::@@CLASS( void )
{
    /*@@END_COMMENT*/

    buildDF( &DF0, string("DF0"), gain0,
             int( NumZeros0+0.5 ), int( NumPoles0+0.5 ),
             zero00, zero01, zQ00, zQ01,
             pole00, pole01, pQ00, pQ01 );

    buildDF( &DF1, string("DF1"), gain1,
```

```
int( NumZeros1+0.5 ), int( NumPoles1+0.5 ),  
zero10, zero11, zQ10, zQ11,  
pole10, pole11, pQ10, pQ11 );
```

```
DF0Copy = DF0;  
DF0Copy.setName( "DF0Copy" );
```

```
/*@@BEGIN_COMMENT*/  
}  
/*@@END_COMMENT*/
```