Algorithmic Composer Program Description

Angelo Fraietta

Student Number 95035879

University of Western Sydney

Music Thesis or Performance - Subject 33460

23 November, 1998

I designed *Algorithmic Composer* to meet a personal need as both a composer and performer of electronic music. I required a tool for the PC platform that would enable me to send MIDI data, which I was manipulating during a performance, to an external sound engine.

I was introduced to algorithmic compositional techniques through the "Max" software package when studying Music Technology 3 in 1996. I used Max for all my performance projects; however, two aspects of Max disappointed me – it was unavailable on most of the computers in the multimedia lab due to its price, and the program was not available for the PC platform. I commenced using C-Sound in Music Technology 5. Real time manipulation and sound generation was not available for the Windows® or Dos environments, but was available for Linux. The disadvantage with C-Sound was that it was both a non-graphical environment, and I was unable use external sound engines. I required a program that would run on PC and allow MIDI input and output; and because I could not buy such a software package, I decided to write one. I set the following specifications for the program:

1. The program must include a graphical user interface (GUI) that enabled composers, who mostly are not trained programmers, to create complex algorithmic structures without requiring a deep understanding of computer science.
2. The program had to enable and encourage documentation. After performing a significant amount of MAX programming, and helping many students with their MAX patches, I realized that patch documentation is necessary to enable patch maintenance or assessment.[1]

---

[1] Nell Dale and Chip Weems, *Introduction to Pascal and Structured design*, fourth edition (Boston: Jones and Bartlett, 1997), p. 7.

3. Patches must be navigable to facilitate patch de-bugging. The user must be able to access and edit the data in an object and all other objects connected to the selected object. The user can thereby navigate through a patch and examine the logic flow through the internal data.
4. The performer must be able to reset the patch to a known condition to enable effective rehearsal of pieces.

I chose the C++ language because it supports object orientated, object based, and procedural programming paradigms[2]. Additionally, C++ will facilitate porting *Algorithmic Composer* to other platforms, including Macintosh and Linux. I planned to use Borland C++ Builder 3.0, a rapid application development (RAD)[3] package, for the Win32 GUI. Builder 3.0, however, was unavailable in Australia until May 1998, so I commenced programming with Borland C++ 4.5 until Builder 3.0 became available. Builder 3.0 arrived with a complimentary C++ 5.02 programming suite. I chose to use C++ 5.02 instead of Builder 3.0 during the construction of the underlying data structures for several reasons. C++ 5.02 had the same integrated development environment (IDE) as C++ 4.5, which enabled me to continue working on the program without immediately having to use a foreign IDE. I had already created a menu driver that used a console interface for the data structures, and therefore did not immediately require a GUI for the program. C+5.02 also had Code Guard, which enabled me to find memory leaks, access violations, and other bugs in my program. The program, however, would not immediately compile due to the changes in the C++

---

[2] Stanley B. Lippman and Josée Lajoie, *C++ Primer*, third edition (Reading, Massachusetts: Addison-Wesley, 1988), p. 2.

[3] Kent Reisdorph, *Teach Yourself Borland C++ Builder in 14Days*, Indiana: SAMS, 1998).

language between the release of Borland C++ 4.05 and C++5.02,[4] albeit only minor changes. I had trouble migrating from C++ 5.02 to Builder 3.0 because Builder 3.0 did not support the class container types from C++ 5.02. This was not a great problem because similar container types, including map and vector, are now included in the C++ standard library.[5] I did, however have to rewrite large sections of code.

I had a great deal of trouble with the timer events, due to the inability of Windows® 95 to process multimedia timers in 32 bit processes. Successful implementation of multimedia timers in Windows® 95 required that the time critical sections of the program exist in a 16-bit dynamic link library (DLL).[6] Flat thunks are then required to allow 32-bit applications access the data structures within the 16-bit DLL.[7] Paul Messick, in his book *Maximum MIDI: Music Applications in C++*,[8] provides a pair of DLLs, called the Maximum MIDI Toolkit, and complete source code; his book, however, only supports implementation with Microsoft® Visual C++. I had to create another DLL to interface the Maximum MIDI Toolkit with the Borland compilers because the import library for the 32-bit side of the DLL pair was in Common Object File Format (COFF) and is not supported by Borland[9].

---

[4] Stanley B. Lippman and Josée Lajoie, *C++ Primer*, p. xviii.

[5] Ibid., p. xix.

[6] Mark McCulley, "Overcoming Timer-Latency Problems in MIDI Sequencers," *Microsoft Software Development Network Online Library* (1996): http://premium.microsoft.com/isapi/devonly/prodinfo/msdnprod/msdnlib.idc?theURL=/msdn_mlatency.htm.

[7] Paul Messick, *Maximum MIDI: Music Applications in C++* (Greenwich CT: Manning, 1988), p. 37 – 38.

[8] Ibid.

[9] Inprise newsgroup, borland.public.cpp, 20 September 1998.

I initially used the synchronizer from the Maximum MIDI Toolkit only to discover that my part of the program that received the timing messages from the toolkit used the Windows Message queue, which was worse than using the multimedia timers in a 32-bit process. I realized at this point that the object data structures in my program needed to be in a 16-bit DLL for accurate timing. I attempted using the thunk compiler as described by Paul Messick, only to find that thunk32.lib on the Microsoft® Software Development Kit (Win32 SDK) was also in COFF format. Borland included the thunk compiler in the C++ 5.02 Development Suite, but did not include any way of implementing it with any Borland compilers. It is possible, however, to create flat thunks using Borland Turbo Assembler (TASM). TASM32 is included with Builder 3.0, but the examples for creating flat thunks are only included with TASM 5.0,[10] which is TASM32 in a separate box and a price tag of $150. After spending three solid weeks attempting to implement flat thunks in my program without the specific TASM example, I decided to use Windows® NT because it uses a true 32-bit multimedia system.[11] The program runs in Windows® 95 with limitations on timing accuracy.

I separated the program into three distinct program groups: MIDI interface, program engine, and user interface. The current implementation of Algorithmic Composer uses the Maximum MIDI Toolkit for MIDI input and output, the timing algorithms and object types exist in the engine, and the main Windows® program is the user interface.

---

[10] I received an example of flat thunking on 17 November 98 but have no time to implement it into my program before this submission is due.

[11] Mark McCulley, "Overcoming Timer-Latency Problems in MIDI Sequencers," *Microsoft Software Development Network Online Library* (1996).

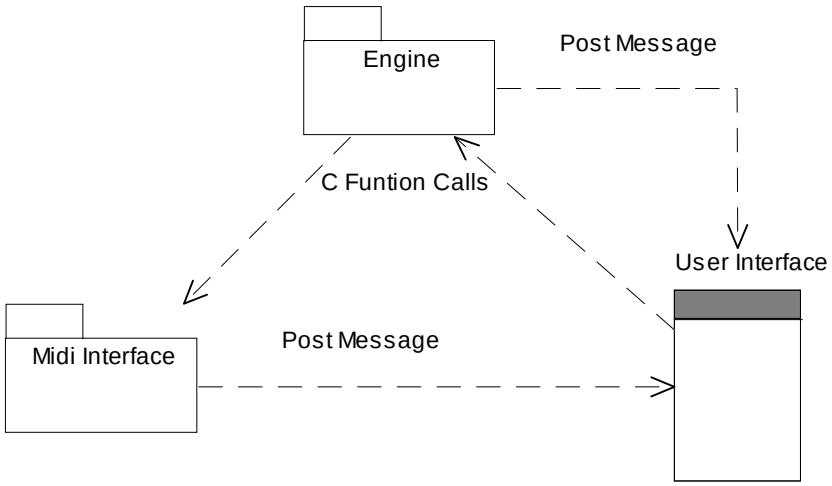

Figure 1 Package Layout

The MIDI interface and Engine exist as a separate DLLs, which will facilitate porting to new platforms and operating systems. Packages are linked with standard C function calls and the PostMessage procedure. The PostMessage procedure sends messages to the user interface through the windows message queue. The PostMessage procedure performs a reentrant call, through the main program at time critical sections of program execution, allowing the program to process the procedure call when the central processing unit (CPU) is idle.[12] An equivalent reentrant procedure or function call would be required when porting the program to a new platform if communication from the engine to the user interface is required. The PostMessage functions inform the user interface of Trigger events, Display events, and error messages, and thus prompt the interface to update the appropriate windows through function calls to the engine. An interface, however, can still operate effectively without this type of communication because the updating of edit windows and icons is achieved by poling the objects when the CPU is idle.

---

[12] Charles Petzold, *Programming Windows 95*, fourth edition (Redmond: Microsoft Press, 1996), p. 46.

The standard C function calls allow platform independence and language portability. A software developer can construct another user interface using a different language, providing the new language supports C function calls. For example, the separation of the engine from the interface will potentially allow ensemble performances via a network. One machine could have the engine loaded into memory while other machines simultaneously access the underlying data structures. The user interface could be written in JAVA, thus creating an Internet ensemble.[13] That area of research, although beyond the scope of this project, could use the engine package of *Algorithmic Composer* as a central element, or simply as a plug-in, without re-compiling the source.

Engine Construction

The engine was constructed using mainly the BaseShell and the Connector classes. Both classes are inherited from Identity class, which has both a name and comment, to enable and encourage patch documentation. (See figure 2).

BaseShells are the object types -- like a counter or delay (see figure 3) -- and are connected together to form a Patch, similar to the way "older analogue machines … used patch cables to connect different electronic modules"[14] together. BaseShells use their common ancestry to connect and communicate. The functions used to connect the object are non-virtual, as the operations they perform are on the BaseShell part of the object.

---

[13] Alex Cockburn had the idea of an Internet ensemble as a project earlier this year.

[14] Jeff Pressing, *Synthesizer Performance and Real-Time Technique (Oxford: Oxford University Press, 1992),* p. 16.

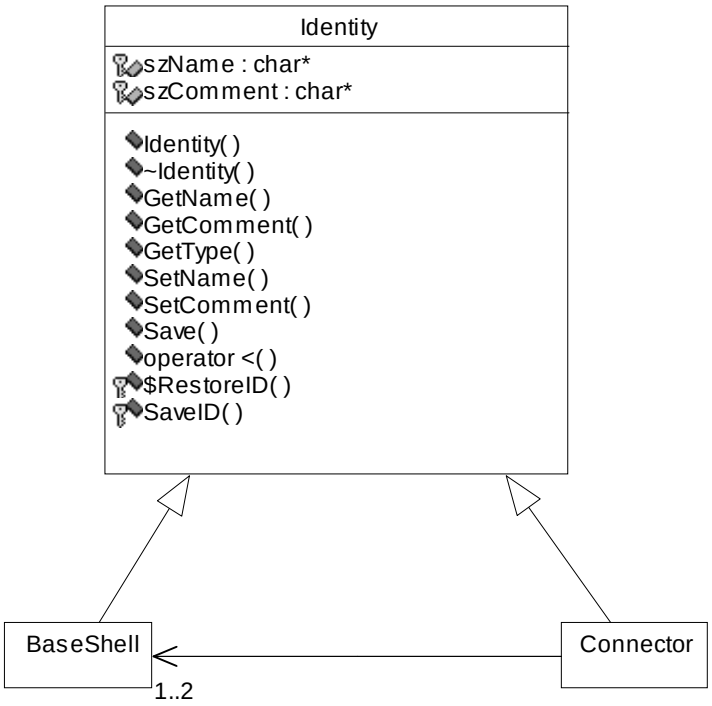

Figure 2 BaseShell and Connector Inheritance

BaseShells and Connectors exist together inside a single Patch, which itself is a type of BaseShell. Each BaseShell and Connector belong to one Patch, whereas a Patch can have many BaseShells or Connectors. The only circumstance where a BaseShell does not belong to a Patch is when the BaseShell is a Patch and not a sub-patch. An example of this instance is when the user creates a new Patch from the file menu. (See figure 4).

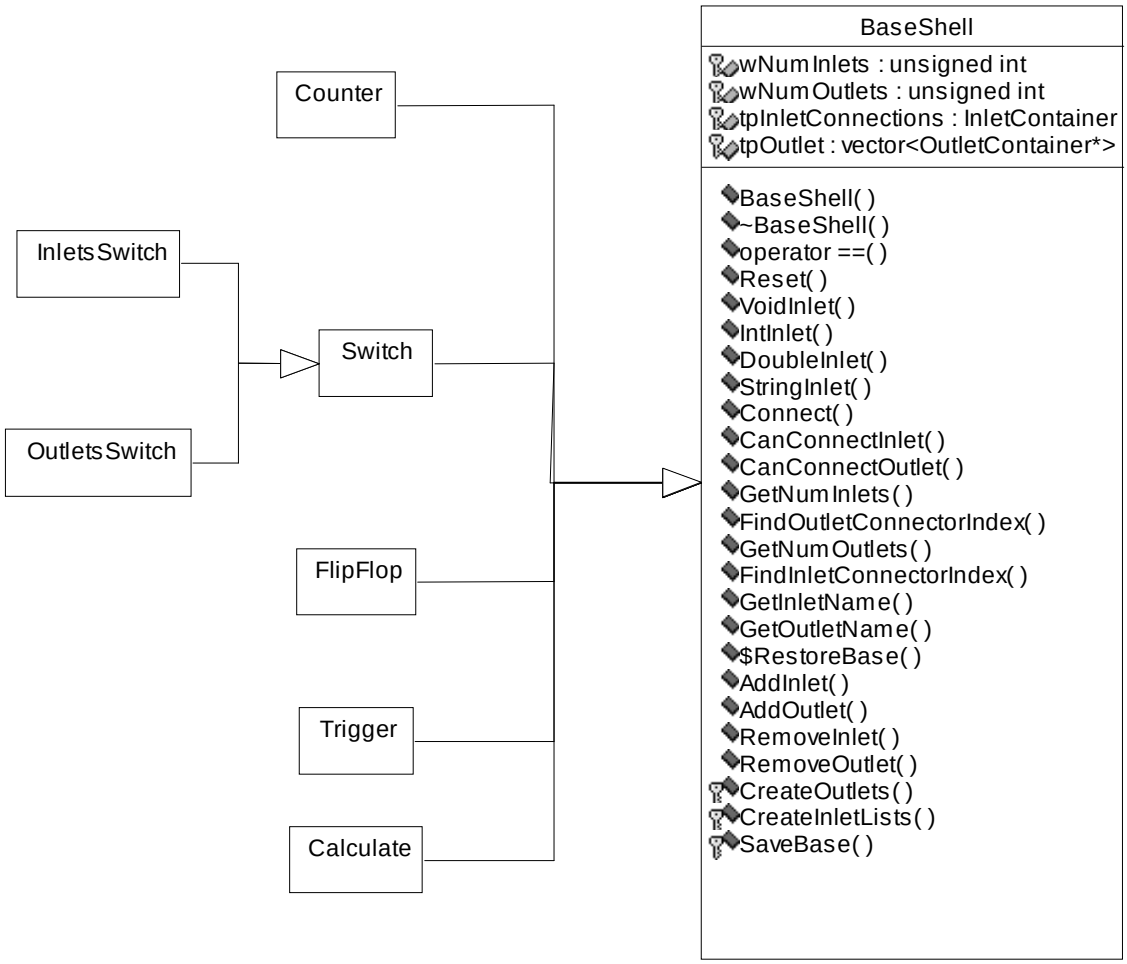

Figure 3 BaseShell Sub-Set Inheritance.

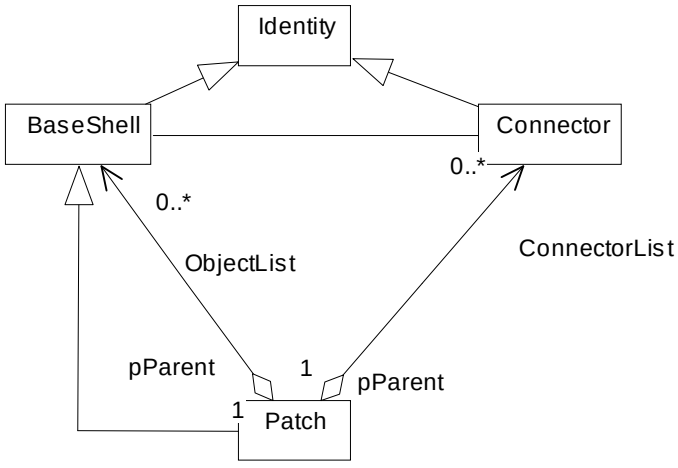

Figure 4 Patch Construction.

A Connector is associated with one or two BaseShells through the Socket Details attributes. (See figure 5).

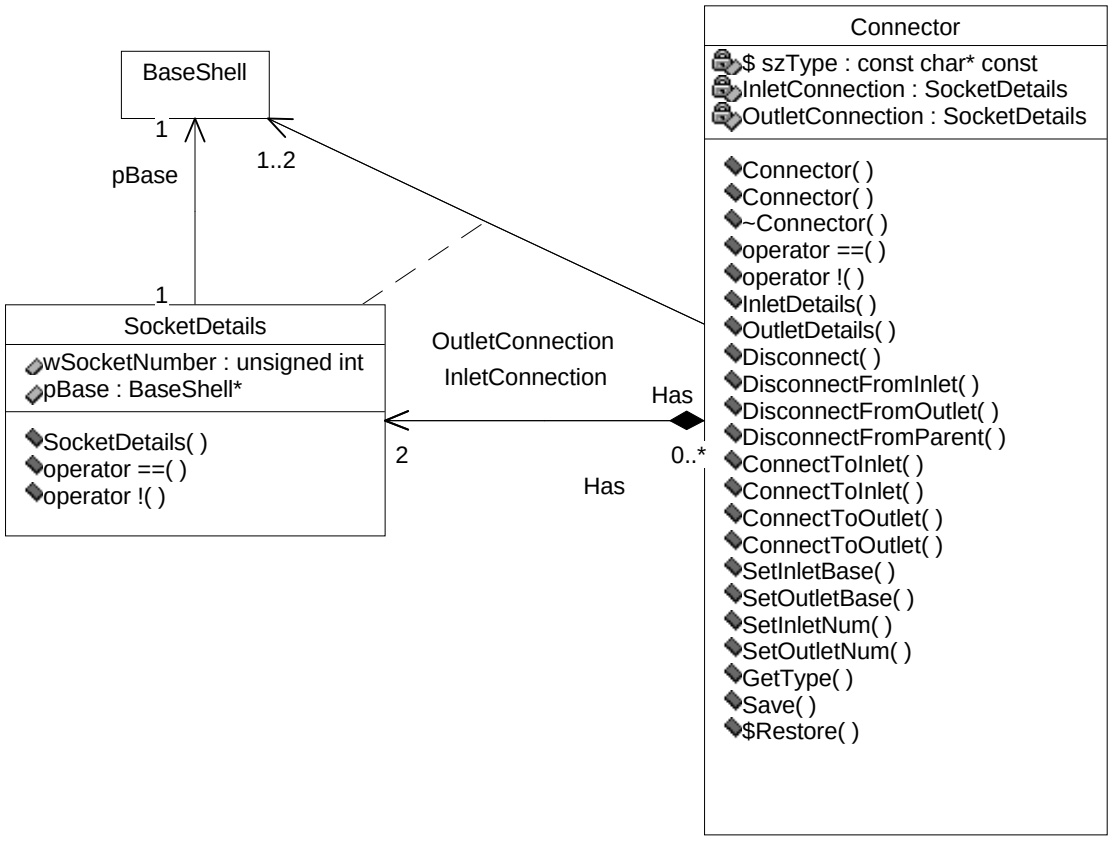

Figure 5 Connector to BaseShell Association

BaseShells are able to call the BaseShell through the Connector by obtaining the address of the other BaseShell, from the Connector's pBase attribute, and the inlet or outlet number from the wSocketNumber attribute. The BaseShell's outlet has an array of Connectors (see figure 6). When an object needs to send a message from an outlet, it iterates through its list of Connectors and uses the attributes from each Connector.

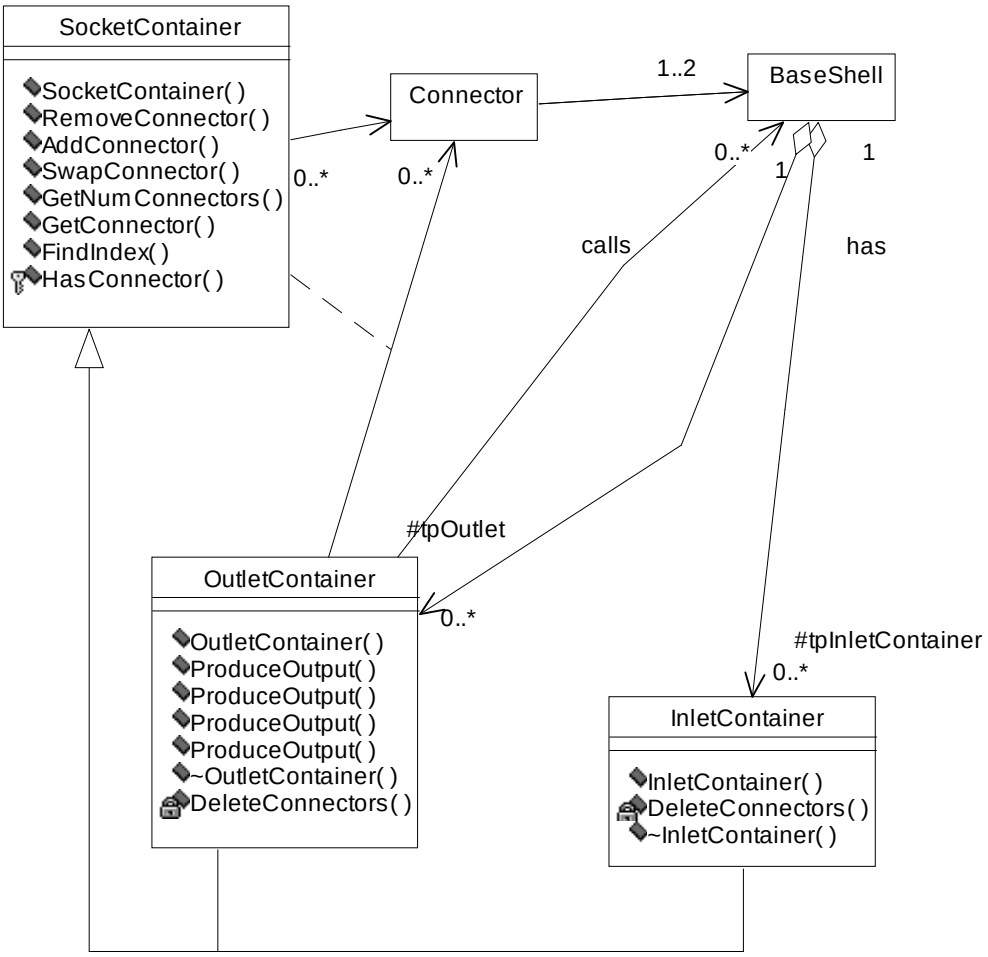

Figure 6 Inlet and Outlet Containers

Messages are sent to an object by using the inlet number as a parameter to the virtual inlet function call to the BaseShell. BaseShell has an array of OutletContainers that have four ProduceOutput methods, and, BaseShell also has four virtual inlet function calls, one for each type of message. The outlet number is defined by the tpOutlet index. The ProduceOutput functions appear as follows:

```
class OutletContainer :public SocketContainer
{
public:
        //other class functions
        ……
        void ProduceOutput();
        void ProduceOutput (const char* sVal);
        void ProduceOutput (int iVal);
        void ProduceOutput (double dVal);
};
```

If a function has to send a message out through an outlet, the outlet number is used as the index to access the appropriate outlet from tpOutlet. The overloaded ProduceOutput function is called on the tpOutlet vector element.

```
void Delay::SendMessage (MessageCell& TheMessage)
{
  switch (TheMessage.MsgType){
    case MessageCell::VoidType:
      tpOutlet[0]->ProduceOutput();
      break;
    case MessageCell::IntType:
      tpOutlet[0]->ProduceOutput(TheMessage.ival);
      break;
    case MessageCell::DoubleType:
      tpOutlet[0]->ProduceOutput(TheMessage.dval);
      break;
    case MessageCell::StringType:
        tpOutlet[0]->ProduceOutput(TheMessage.sval.c_str());
        break;
    default:
      break;
  };//end case
};
```

The Inlet function prototypes appear in the BaseShell class definition as follows:

```
class BaseShell: public Identity{
public:
        //other class functions
        ……
        virtual void VoidInlet (unsigned InletNumber) {}
        virtual void IntInlet (unsigned InletNumber, int iVal){}
        virtual void DoubleInlet (unsigned InletNumber, double dVal){}
        virtual void StringInlet(unsigned InletNumber, const char* sVal){}
        //other class functions
};
```

The functions are not pure virtual functions, and as such, each subset object only needs to overload the virtual functions it requires. The function call is not made to the InletContainer, but rather to the virtual function. The main purpose of tpInletContainer is to notify the Connectors pointing to this object when this object is

about to be destroyed, thus preventing dangling pointers.[15] The Inlet Number is used as an index to an array of function pointers that are object specific. Consider the following Delay class:

```
class Delay: public BaseShell, DelayLine
{
public:
        //other functions
        …..
        //overloaded inlet functions
        void VoidInlet(unsigned);
        void DoubleInlet(unsigned, double);
        void IntInlet(unsigned, int);
        void StringInlet(unsigned, const char*);


         void Reset ();
         void SetInterval(int i){wInterval = (unsigned)i;}
        //other functions
private:
        //other attributes and functions
        ……
        //these are the private functions that are called from inside Delay
        void DelayVoid();
        void DelayInt(int);
        void DelayString(const char*);
        void DelayDouble(double);
        void StringInlet1(const char*);

        //these are the static arrays of pointers to functions
        typedef void(Delay::*pVoidFunc)();
        static const pVoidFunc tpVoidFunc[];

        typedef void(Delay::*pIntFunc)( int);
        static const pIntFunc tpIntFunc[];

        typedef void(Delay::*pDoubleFunc)(double);
        static const pDoubleFunc tpDoubleFunc[];

        typedef void(Delay::*pStringFunc)(const char*);
        static const pStringFunc tpStringFunc[];

};
```

[15] Stanley B. Lippman and Josée Lajoie, *C++ Primer*, pp. 406-09.

The statement:

```
typedef void (Delay::*pIntFunc)(int);
```

defines pIntFunc as a pointer to a Delay class member function that takes one integer argument and a return type of void. Therefore

```
static const pIntFunc tpDoIntFunc[];
```

declares a constant static member that is an array of type pIntFunc. The array is then defined:

```
const Delay::pIntFunc  Delay::tpIntFunc [NUM_INLETS] = {
        &Delay::DelayInt,
        &Delay::SetInterval
};
```

Returning to the Delay class definition we see that DelayInt and SetInterval are both functions that take one integer as an argument and return void. Delay::tpIntFunc[1] gives the address of Delay::SetInterval. If no function is desired for the input, the function address in the static array is set at NULL.

```
const Delay::pDoubleFunc  Delay::tpDoubleFunc [NUM_INLETS] = {
        &Delay::DelayDouble,
        NULL
};
```

Implementing the call to IntInlet of the Delay is as follows:

```
void Delay::IntInlet(unsigned InletNumber, int i)
{
        if ((InletNumber < GetNumInlets())&& (tpIntFunc[InletNumber]))
                (this->*tpIntFunc[InletNumber])(i);
}
```

The index is first checked, InletNumber < GetNumInlets(), and then the function address is taken checked for validity with InletNumber used as the index. If both conditions are met, the function is executed.

```
void Delay::DoubleInlet(unsigned InletNumber, double d)
{
        if ((InletNumber < GetNumInlets())&& tpDoubleFunc[InletNumber]))
                (this->*tpDoubleFunc[InletNumber])(d);
```

```
}
```

A double message at Inlet 1 will not execute because Delay::tpDoubleFunc[1] is equal to NULL. The same method of pointer to class member function is used to acquire the names to the inlets and outlets. A static array of const char* const is placed in each object and is accessed the same way through the virtual BaseShell functions

```
const char* GetInletName(unsigned InletNumber);
const char* GetOutletName(unsigned OutletNumber);
```

and implemented in the sub-class

```
const char* const Delay::szaInletDetails[NUM_INLETS] = {
        "Message",
        "Interval"
};

const char*const Delay::szaOutletDetails[NUM_OUTLETS] = {
        "Delayed Message"
};

const char* Delay::GetInletName(unsigned InletNumber)const
{
        if (InletNumber < GetNumInlets())
                return szaInletDetails[InletNumber];
        else
                return NULL;
}
```

BaseShell also uses pure virtual functions to access attributes from the sub-class that do not exist in the BaseShell super-class. The function

```
const char* GetType() = 0;
```

in the Identity class definition allows access to the type which is defined statically in the sub-class; and

```
void Reset()  =  0;
```

in the BaseShell class definition allows access to the Reset function. Instantiation of Identity and BaseShell classes is therefore not possible except through derived classes that override these pure virtual functions.

New Objects are instantiated through the sub-set class member static functions

```
static BaseShell* Create(Patch* pParent);
static BaseShell* Restore(const char**,unsigned long*, Patch*);
```

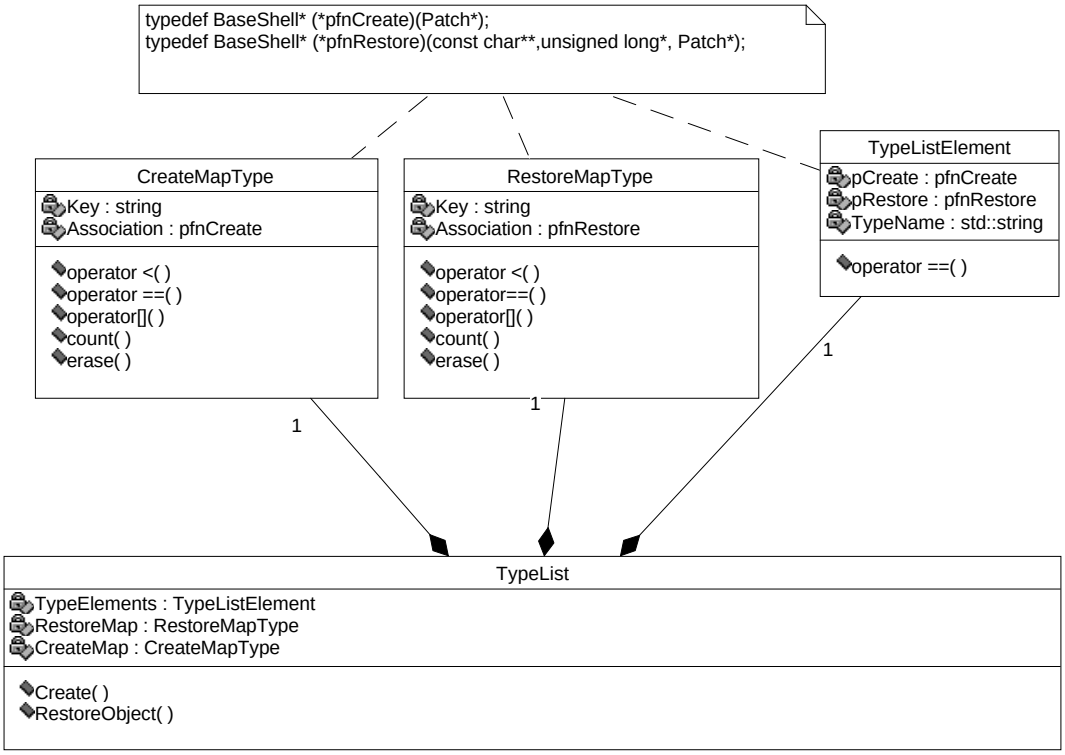

Figure 7 Create and Restore Diagram.

The address of each Create and Restore function and its type is stored within the TypeListElement class with the following type definitions:

```
std::string TypeName;
typedef BaseShell* (*pfnCreate)(Patch*);
pfnCreate pCreate;
typedef BaseShell* (*pfnRestore)(const char**,unsigned long*, Patch*);
pfnRestore pRestore;
```

(See figure 7).

TypeName is a string that defines the object file -- for example, Counter or Delay.

The list of all type name definitions is contained in the file “TypeNames.h” and must

be accessible to the user interface during compilation. The pfnCreate type is an address of a function that takes a Patch* as an argument and returns a BaseShell*, while pfnRestore takes a const char**, an unsigned long*, and a Patch* as arguments, and returns a BaseShell*.

TypeList has a RestoreMap and CreateMap that use the type name as the key and return the pfnRestore or pfnCreate for that object type. This implementation is hidden within the TypeList class, but is accessed through the function calls

```
static BaseShell* Create (const char* Type, Patch* pParent);
```

and

```
static BaseShell* RestoreObject(std::string Type, const char** pszCursor,
                                unsigned long* pOldPointer, Patch* pParent);
```

within the TypeList class definition.

The call to TypeList::Create uses Type as the parameter that defines the type, and pParent as the Parent Patch. TypeList looks inside its CreateMap for an element that matches the Type argument. If a match is found, it retrieves the address of the Create function and calls Create with the pParent as the argument, otherwise the call returns NULL. Consider the following call to create a new Delay:

```
BaseShell* pBase = TypeList::Create("Delay", pPatch);

//enter into TypeList
BaseShell* TypeList::Create(const char* szType, Patch* Parent)
{
//Parent = pPatch
std::string Type(szType); //converted into string type. Type = Delay
if(NewObjectMap.count(Type)) // is "Delay" here ?
   return NewObjectMap[Type](Parent); //ie. return Delay::Create(Parent)

//enter into Delay
BaseShell* Delay::Create(Patch* Parent)
{
        return new Delay (Parent); //let new Delay = pDelay
}
```

```
//return to TypeList
//TypeList returns pDelay
pBase = pDelay
```

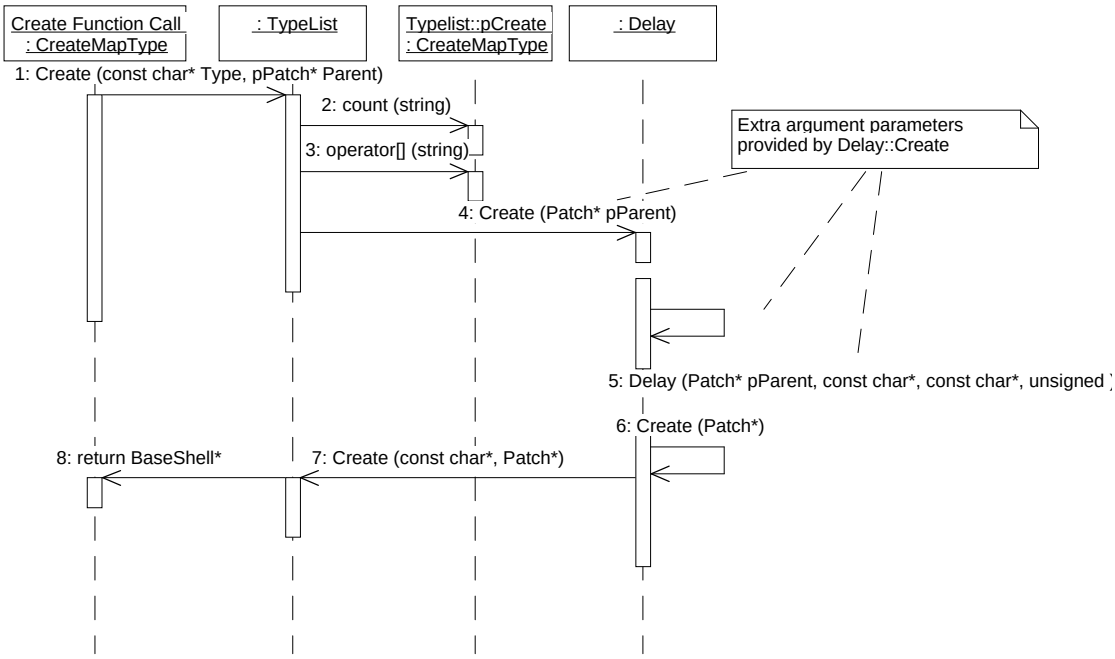

Figure 8 Create Sequence Diagram.

TypeList returns the pointer to a new BaseShell that is actually a Counter. (See Figure 8).

Restore works in a similar manner, restoring a class from file. We must first examine how the data is saved to disk before restoring. The data for each instance of a class is saved as one line of text. Patch calls the virtual Save function for each object in its pObjectList. Each parameter of data is saved as a null terminated string. The Identity part of the object is saved first followed by the BaseShell, and then the class specific data. The object type is saved with the Identity[16], while the pointer is saved with the BaseShell part so the Connectors can find which objects they are connected to.

[16] GetType is a pure virtual function of Identity.

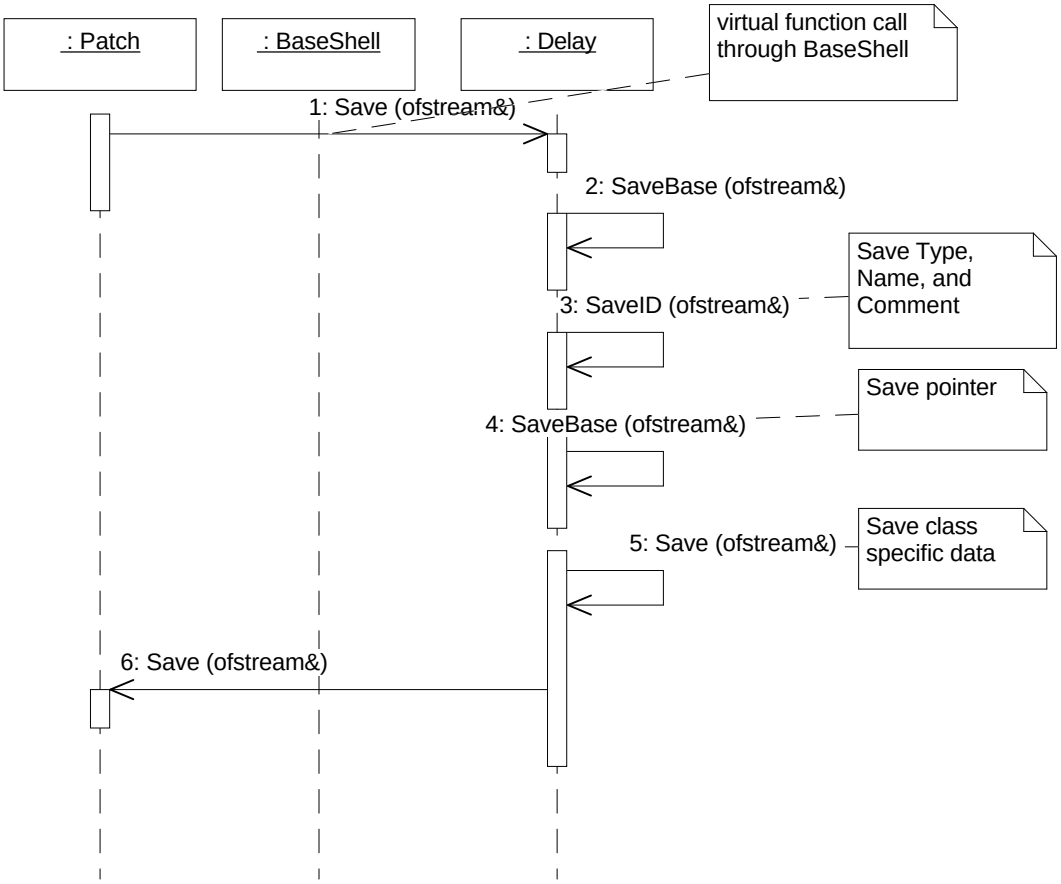

Figure 9 Save Sequence Diagram.

The resultant line of text in the file is

Delay Name Comment 19228988 500

Restoring the objects is done a line at a time, so a character buffer can hold all data associated with the object. Data is restored by Identity first, then BaseShell, followed by class specific data. The data is converted from a character buffer from a line from a file into the appropriate data types string at a time.

```
static BaseShell* RestoreObject(std::string Type, const char** pszCursor,
                                unsigned long* pOldPointer, Patch* pParent);
```

Assuming Type has been found within TypeList, TypeList calls the static Restore function for the particular class Type. The argument pszCursor is a pointer to the character buffer, pOldPointer is the address where the old pointer value will be

stored in the calling function, and pPatch is the Parent Patch. Consider the Restore function within the Delay class:

```
BaseShell* Delay::Restore(const char** pszCursor,
        unsigned long* pOldPointer, Patch* Parent)
{
   const char* szName;
   const char* szComment;
   *pOldPointer = RestoreBase(pszCursor, &szName, &szComment);

   unsigned riInterval = (unsigned)atoi(*pszCursor);
   *pszCursor = *pszCursor + strlen(*pszCursor)+1;

   return new Delay(Parent, szName, szComment, riInterval);
}
```

RestoreBase assigns szName, szComment to positions along the character buffer *pszCursor, returns the value of the old pointer, which is assigned to *pOldPointer, and moves pszCursor to the next point in the character buffer after the old pointer. The local variable riInterval is declared and assigned to the converted value of the null terminated string that pszCursor is pointing to, after which pszCursor is moved to the next position. The variables pParent, szName, szComment and riInterval are passed as parameters in the creation of a new Delay. Delay::Restore returns a pointer to the new Delay (as a BaseShell*) to the Patch that is being restored. The Patch stores the real pointer in pObjectList, while the old pointer is stored in a temporary map that associates the old pointer with the new pointer. (See figure 10).

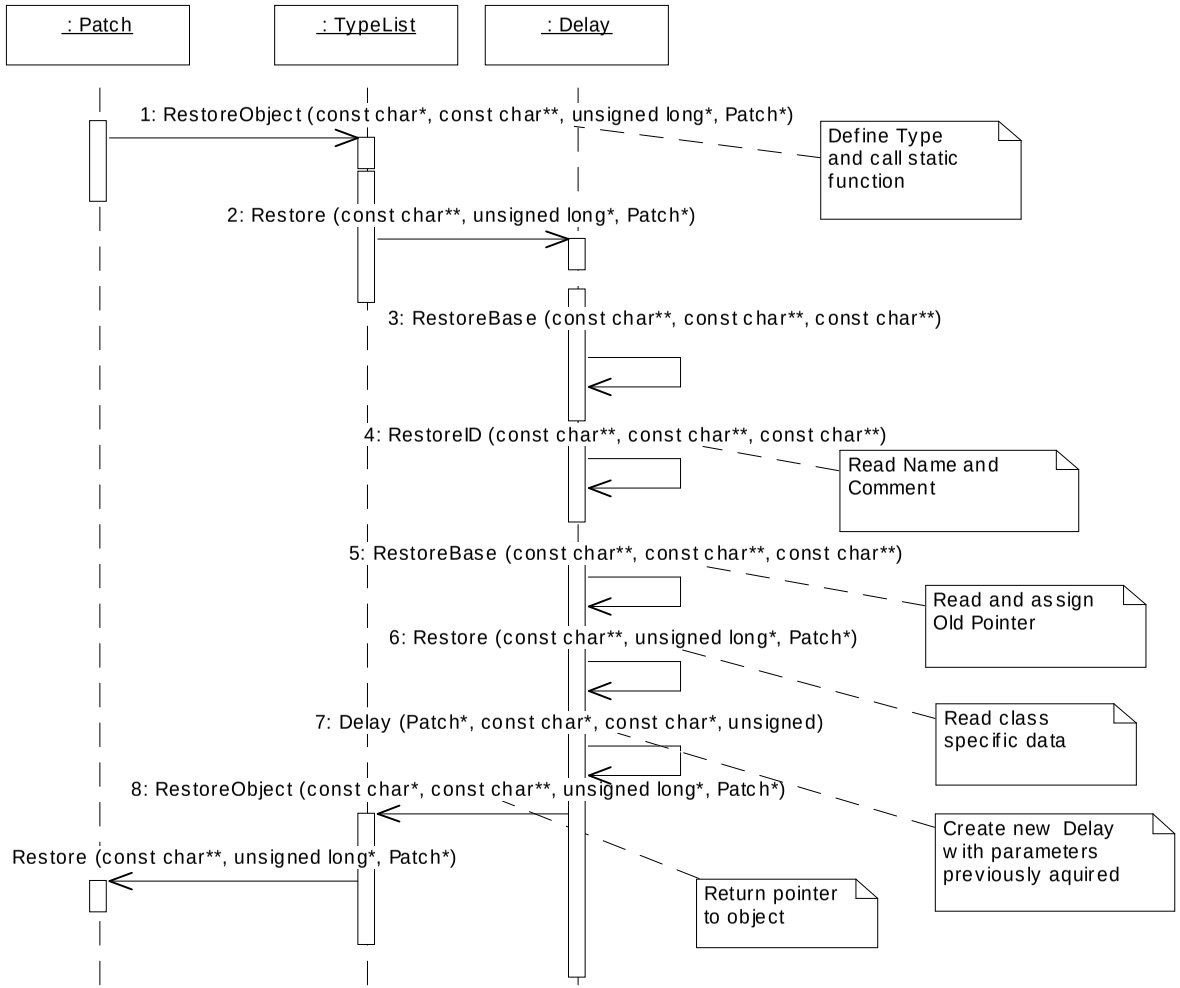

Figure 10 Restore Sequence Diagram.

Connectors are saved in an identical manner, using the pointer to the inlet and outlet objects, and the inlet and outlet numbers as the class specific data. Patch class the function RestoreConnector, which uses the old BaseShell pointers as the key to acquiring the new BaseShell pointers from the map created in Patch::Restore. The Connector is created with the restored Name, Comment, object pointers, and inlet and outlet numbers. The Patch uses the same temporary object map to associate Patch inlets and outlets. (See figure 11).

An example of a patch saved to disk is

```
Patch New Patch File1  19226672 0 0 1
Inlets Switch   19229016 5 0
Outlets Switch   19232392 5 0
Number Store   19234740 0 6 0
Trigger   19236956
Trigger   19239160
Trigger   19241364
Trigger   19243568
Number Store   19245772 0 127 0
Trigger   19247988
Trigger   19250192
Connector   19229016 0 19236956 0 19252424
Connector   19232392 0 19243568 0 19255580
Connector   19232392 4 19247988 0 19257708
Connector   19234740 0 19229016 0 19259836
Connector   19234740 0 19232392 0 19261964
Connector   19239160 0 19229016 1 19221348
PatchPortPointers
End Patch 19226672
```

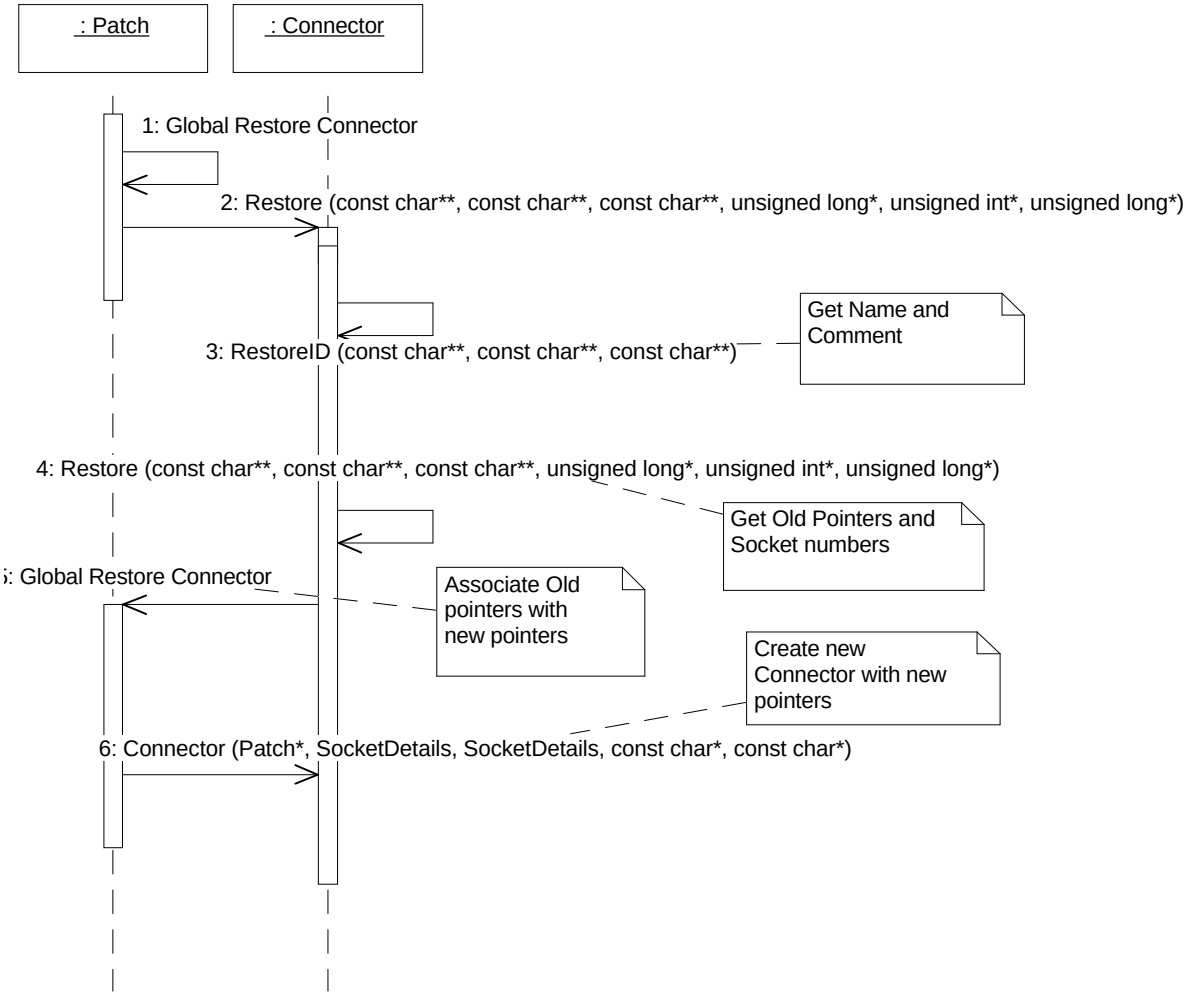

Figure 11 Restore Connector Sequence Diagram.

Object Scheduling

Metro, Delay, and Sequencer use multiple inheritance to enable them to be both BaseShells and TimerObjects (see figure 12). TimerObjects send callback requests to the Scheduler, which implements the timing through the operating system

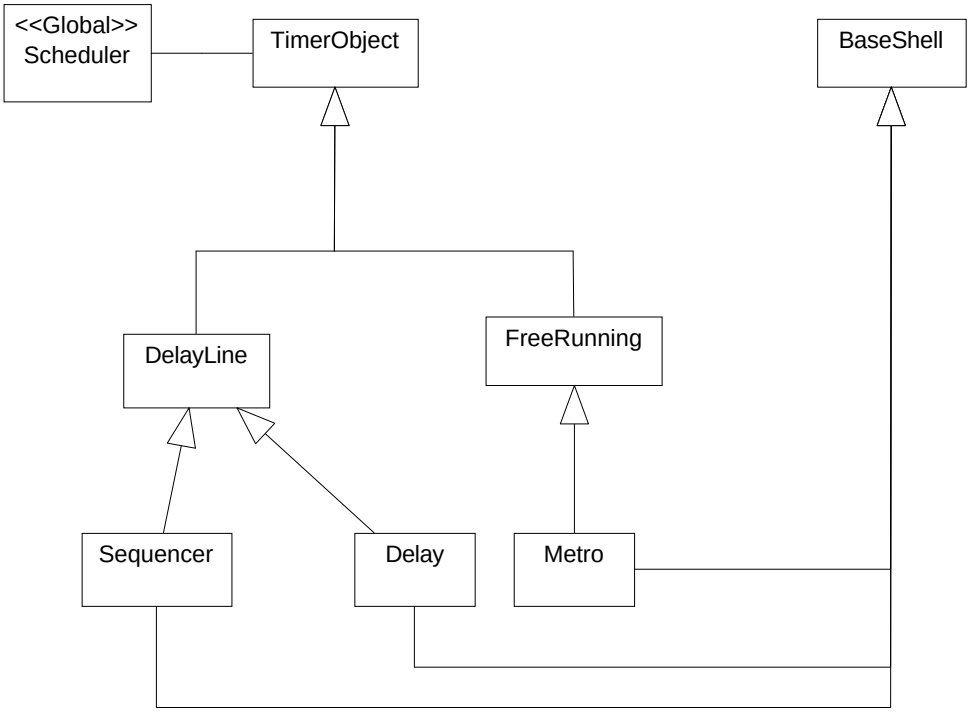

Figure 12 Timing Objects Inheritance Diagram.

A TimerObject gets the current time by calling GetPosition from Scheduler and adds a timing interval to calculate the callback time. The TimerObject sends a TimerQueue item with the pointer to itself and the callback time as attributes (see figure 13). The Scheduler pushes the event, which uses the callback time as the priority, onto the ScheduledItems priority queue. If the new event is at the front of the ScheduledItems priority queue, the Scheduler sends a single callback request to the operating system (see figure 14). The TimerObject calls KillTimeEvent to delete an event. The event is pushed onto the DeletedItems priority queue (see figure 15).

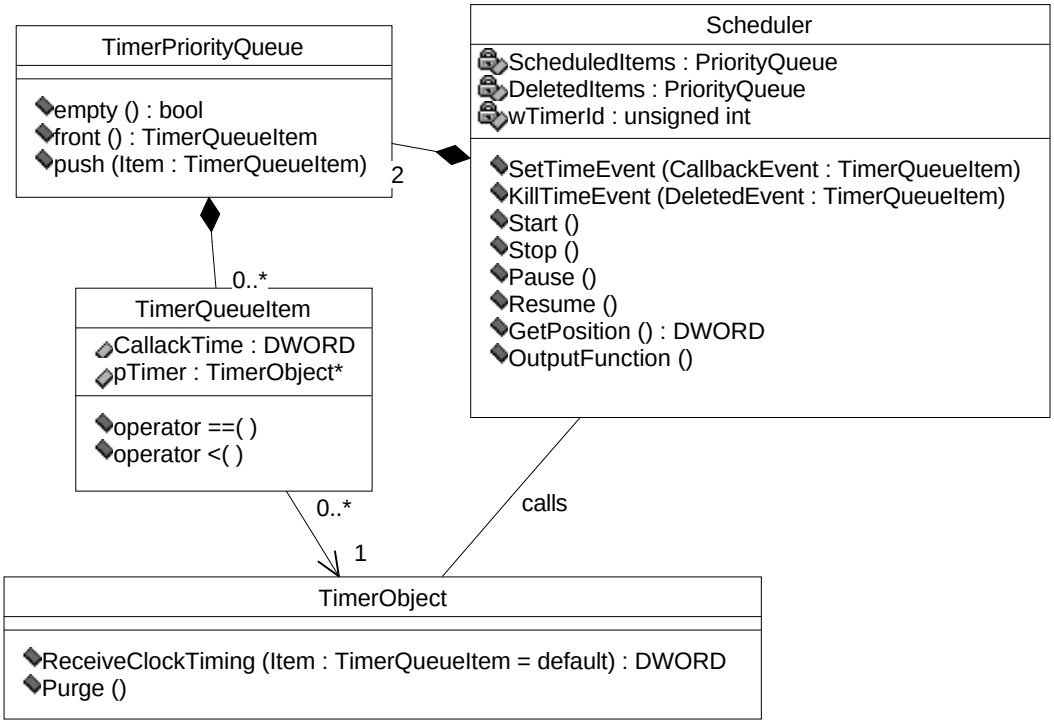

Figure 13 Callback Scheduling.

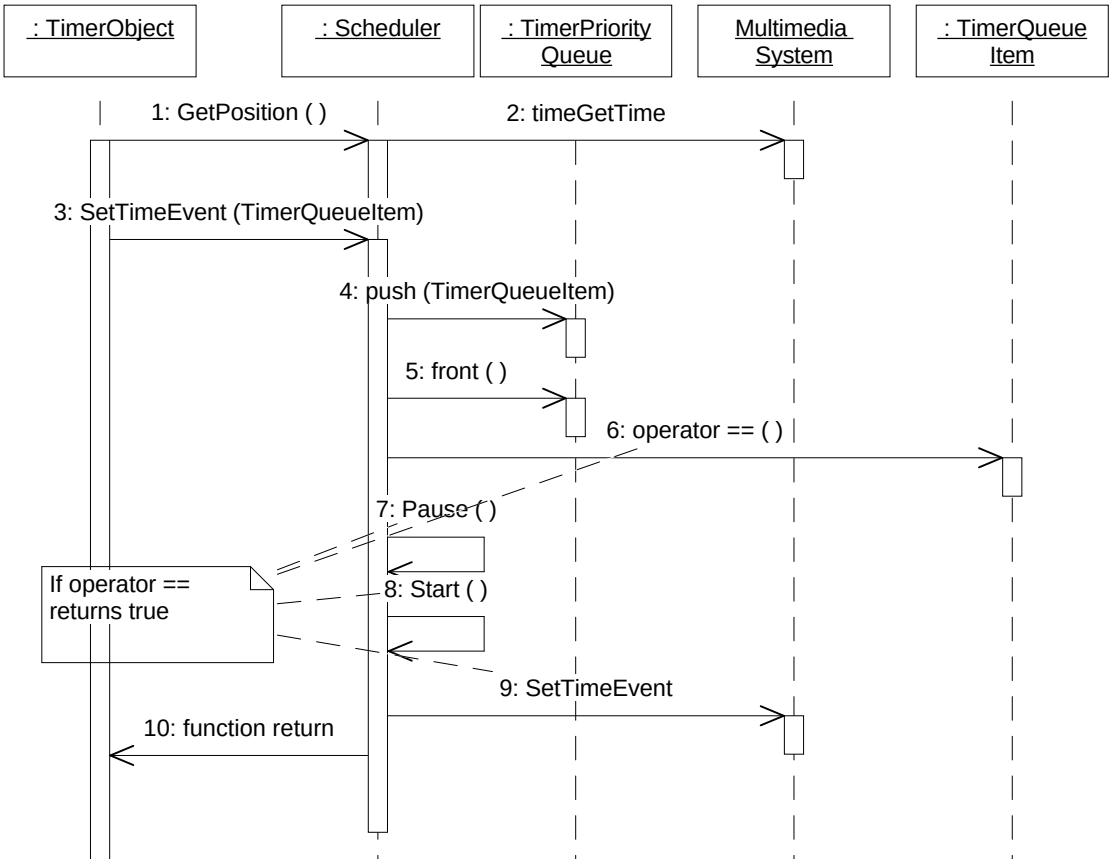

Figure 14 Schedule Set Time Event.

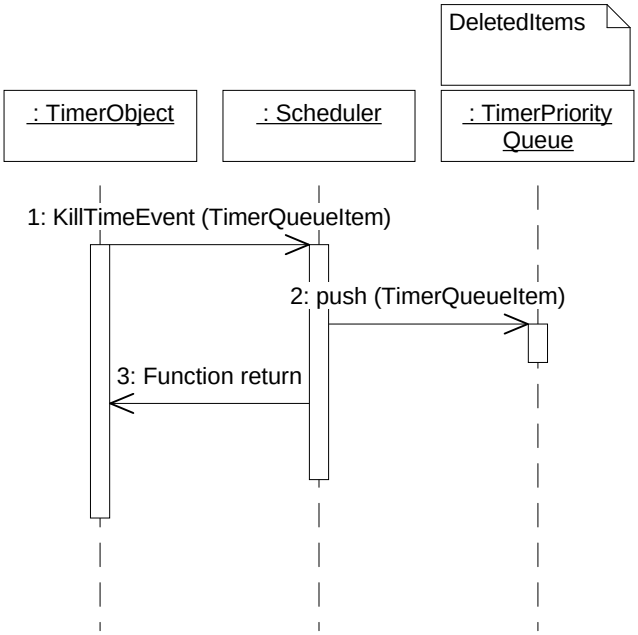

Figure 15 Scheduler Kill Time Event.

The operating system calls the Scheduler at callback time. The Scheduler reads the front of the ScheduledItems queue and, if it is not at the front of the DeletedItems queue, it calls the ReceiveClockTiming function by de-referencing the pTimer attribute of the TimerQueueItem. The function returns with the next requested callback time for this TimerObject, which is in turn pushed onto the ScheduledItems queue by the Scheduler. The Scheduler iterates with this procedure until the callback time of the TimerQueueItem at the front of the ScheduledItems queue is outside the tolerance of the current operating system callback, or the ScheduledItems queue is empty. A callback request is then sent to the operating system if the ScheduledItems queue is not empty (see figure 16).

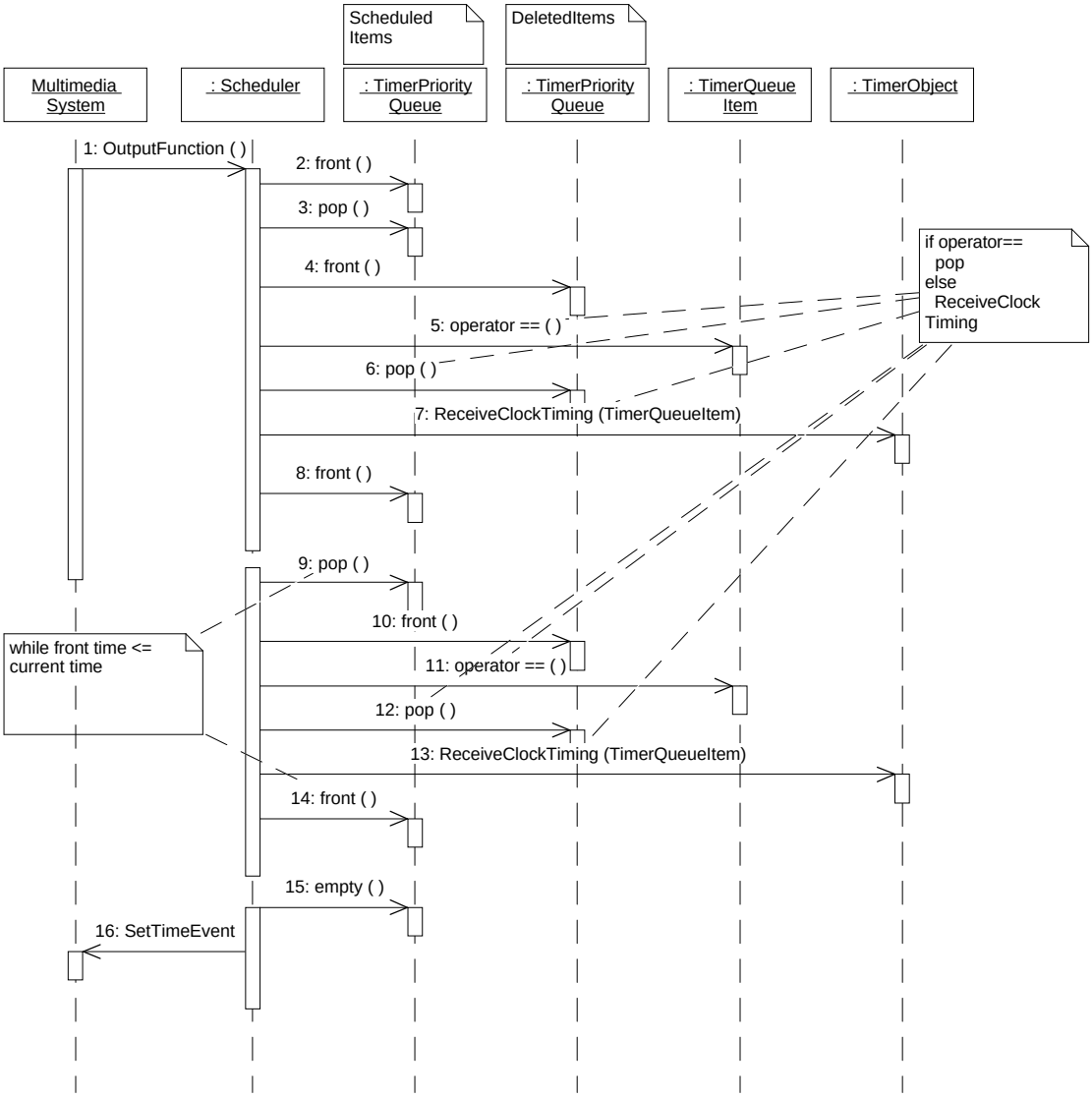

Figure 16 Multimedia System Callback.

FreeRunning holds one event and overloads the virtual function ReceiveClockTiming. ReceiveClockTiming causes FreeRunning to execute the virtual function OutputFunction, which its descendant Metro overloads, causing Metro to produce an output. ReceiveCallBackTime returns the next callback time (see figure 18).

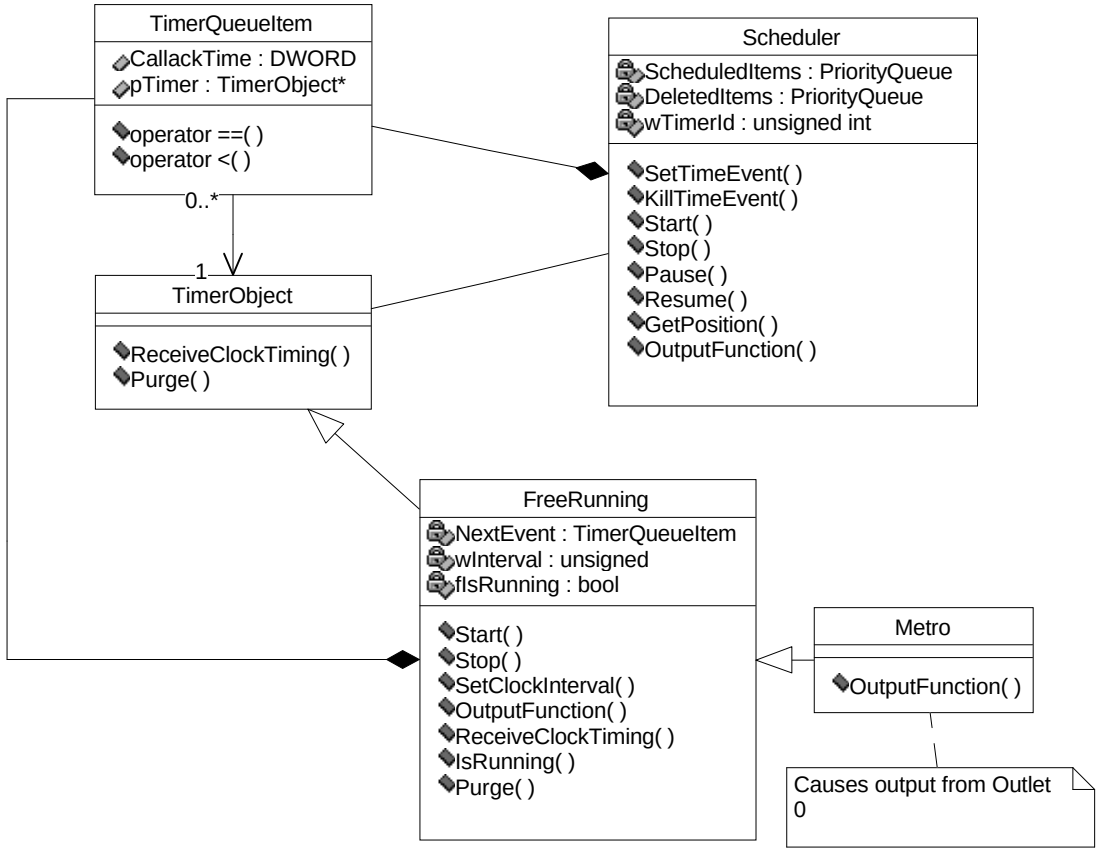

Figure 17 Free Running Class.

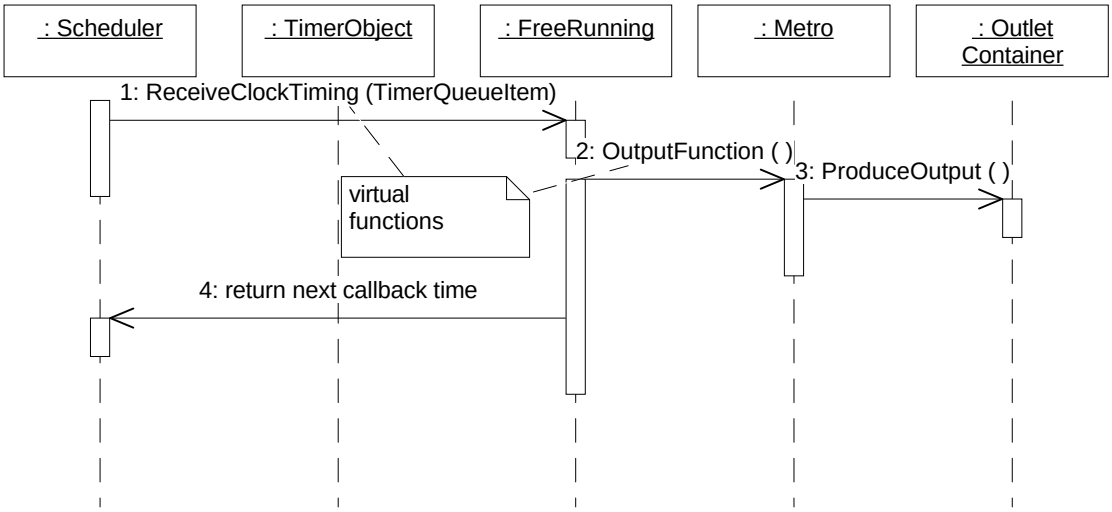

Figure 18 Metro Sequence Diagram.

DelayLine has a priority queue of DelayLineQueueItems that has the DWORD attribute lparam, which is received from the member function call GoOneShot through DelayLine ancestor (see figure 19). When a Scheduler calls

ReceiveClockTiming, the DelayLineQueueItem is taken from the front of the queue, and the lparam value is used as the argument in the OutputFunction, which is overloaded by the sub-class class (see figure 20 and 21). Delay uses the lparam as a pointer to a message cell, whereas Sequencer does not use lparam. Purging DelayLine causes it to call PurgeItem for each item in qParam.

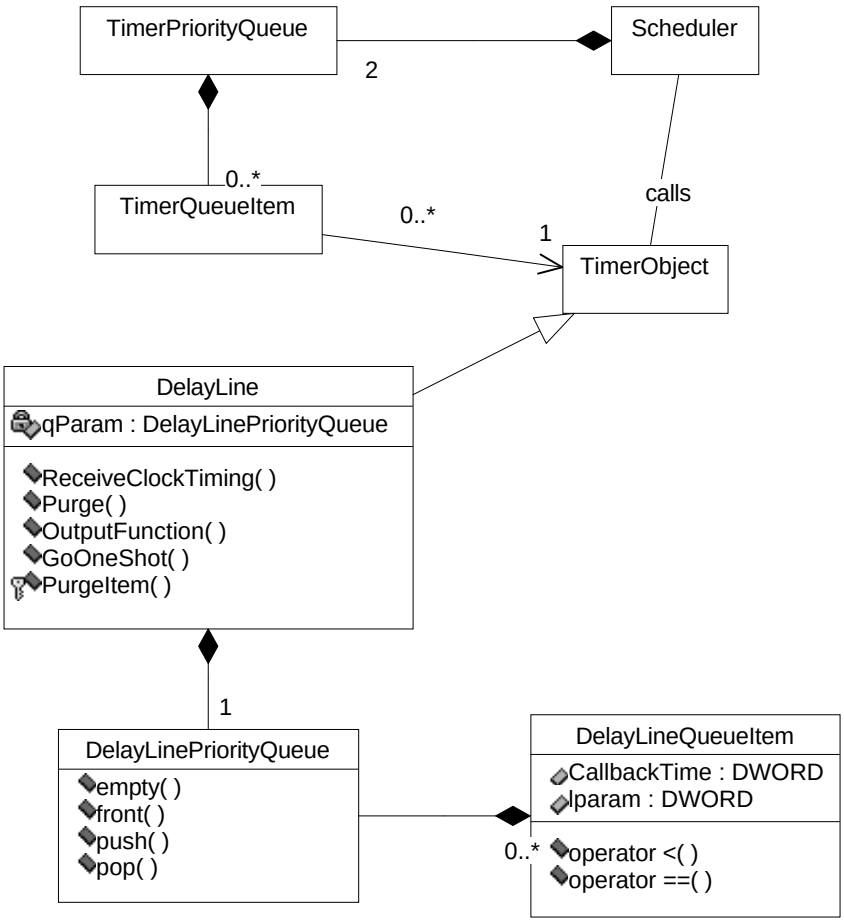

Figure 19 Delay Line Diagram.

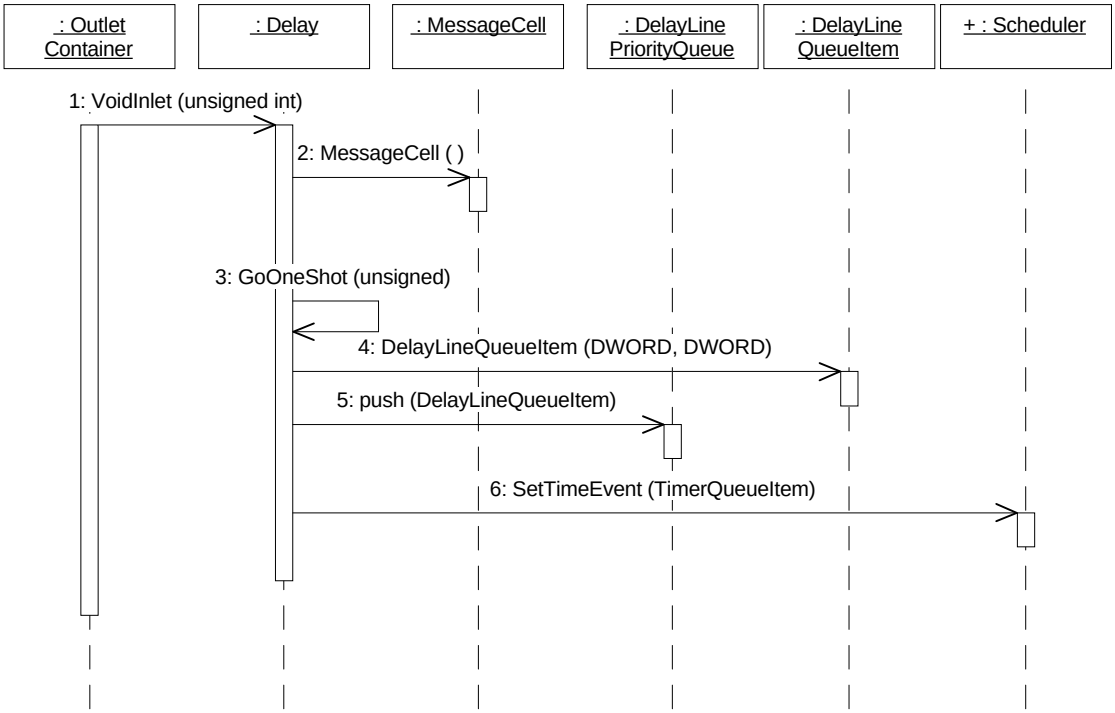

Figure 20 Set Delay Event.

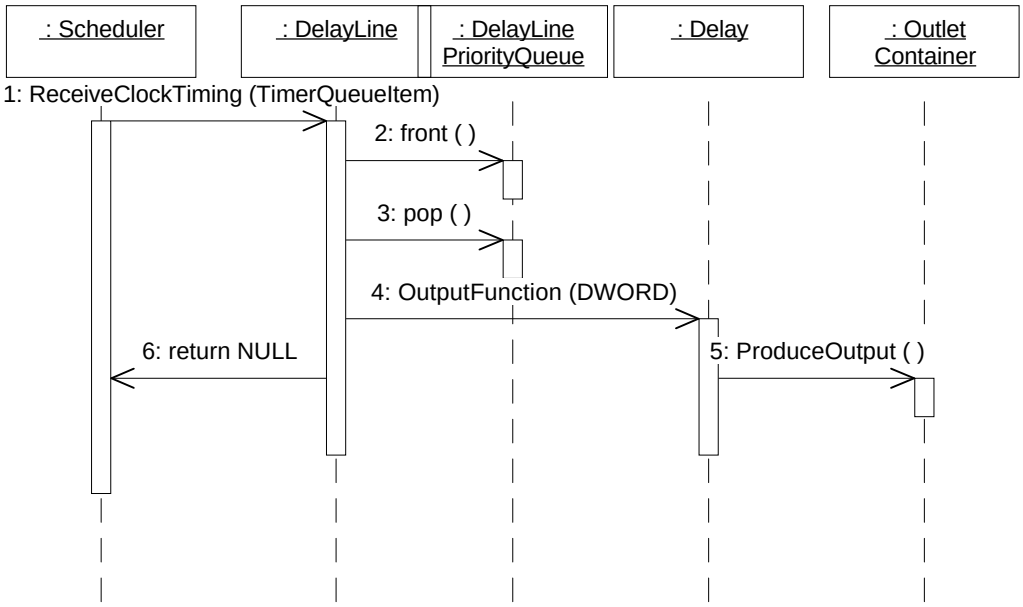

Figure 21Delay Receive Callback.

Engine Bridge.

The user interface receives data from the engine through standard C function calls. Implementing the engine in Windows® 95 for later releases will require the main engine to exist in a 16-bit DLL, which will require thunking with the 32-bit

process. Data alignment in 16-bit processes is different to data alignment in 32-bit processes and so only certain types of data structures can be passed through the thunk compiler.[17] Class structures cannot pass through the thunk layer; therefore pointers to the BaseShell and Connector objects cannot be de-referenced by the user interface. To overcome this limitation, they are converted to DWORDs and passed as a parameter to the function in the Bridge that accesses the BaseShell or Connector (see figure 22). The Bridge casts the DWORD to the appropriate pointer for the function and de-references it within the same DLL. For example:

```
#define DWORD unsigned long;
#define P_BASESHELL DWORD;

unsigned BaseShellGetNumInlets (P_BASESHELL pObject)
{
        return ((BaseShell*)pObject) ->GetNumInlets();
}
```

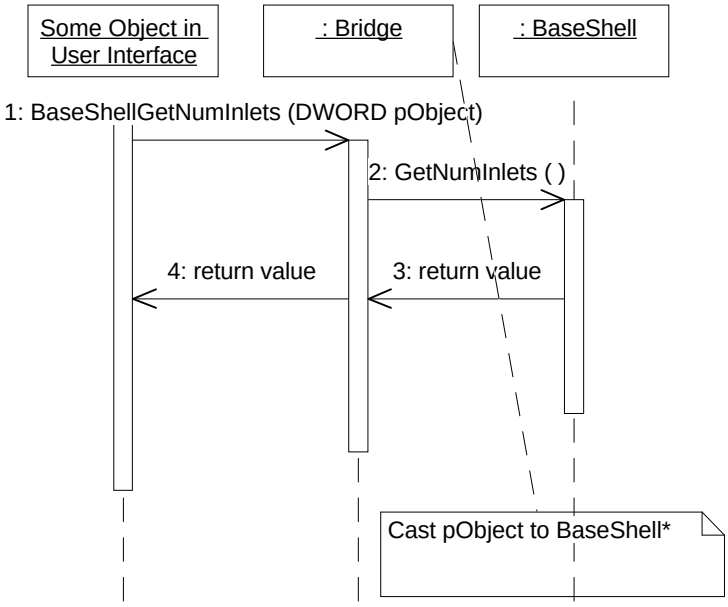

Figure 22 Calling BaseShell through Bridge.

[17] "Handling Data Types Not Supported by the Thunk Compiler," *Microsoft Software Development Network Online Library*: http://premium.microsoft.com/isapi/devonly/prodinfo/msdnprod/msdnlib.idc?theURL=/msdn/library/sdkdoc/win95/tc_2ugj.htm.

Timing messages for the engine would occur during interrupt time, which means the engine cannot call a function to the 32-bit process. The engine posts a message instead, using the window handle of the window in the 32-bit process it needs to communicate with. For example

```
void Trigger::Send()
{
tpOutlet[0]->ProduceOutput();
 if(DisplayWindow)
        PostMessage (DisplayWindow, TRIGGER_MESSAGE,
        (DWORD)this, NULL);
}
```

Header files required to interface to the engine are at Appendix A.

User Interface.

The user interface uses two classes to control memory management within the interface. The TBaseShellHook and the TConnectorHook classes own all the memory for visual component library (VCL) objects associated with BaseShells and Connectors, which in turn are owned by the Patch Form (see figures 23 and 24).

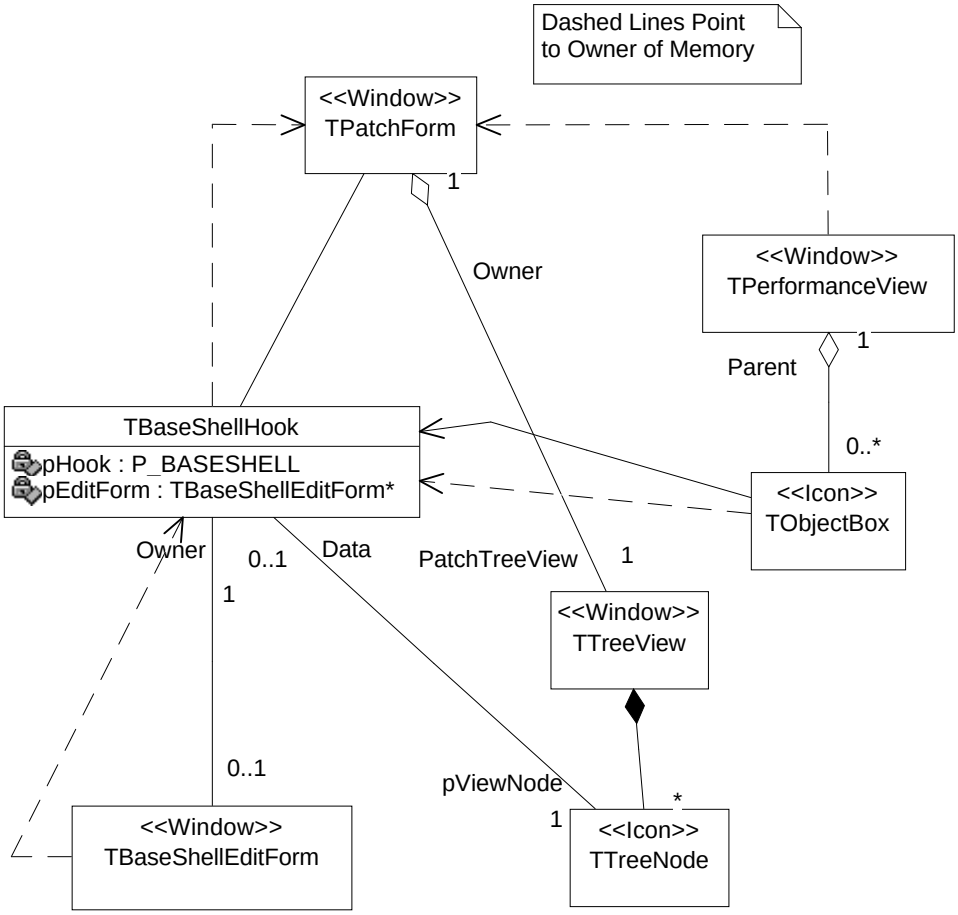

Figure 23 TBaseShellHook Ownership

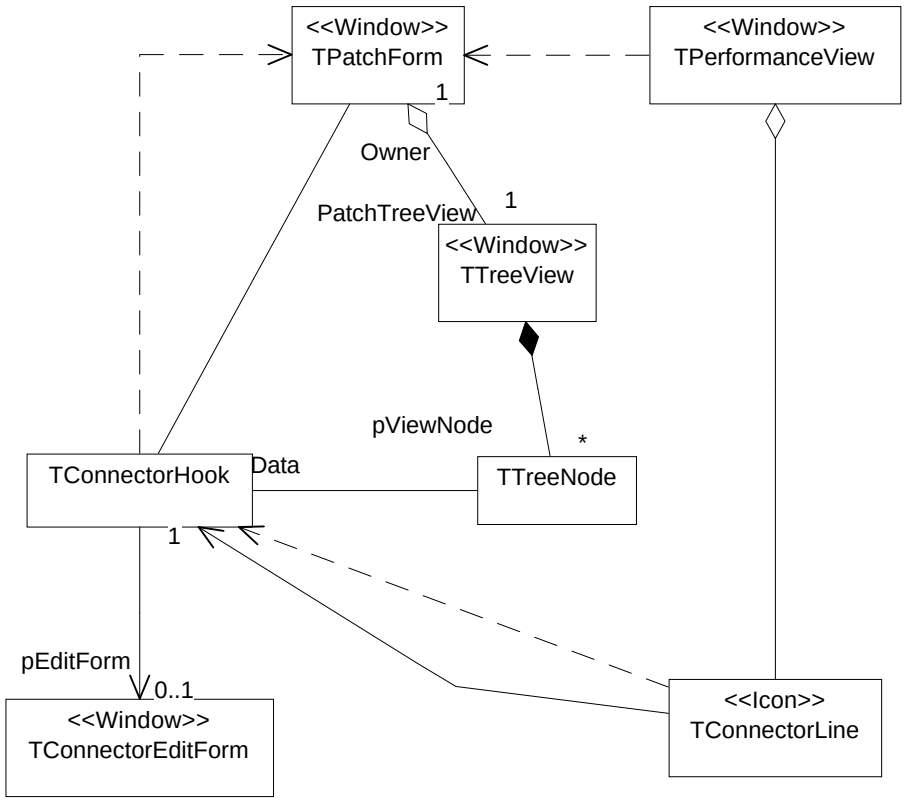

Figure 24 Connector Hook Ownership.

TBaseShellHooks are always deleted through the PatchForm. The TBaseShellHook determines whether it needs to call a delete function upon the BaseShell through the bridge. The TBaseShellHook automatically deletes any windows or icons that are dependent upon it. TConnectorHook operates in the same way.

Invoking an edit window or creating an icon for a particular type of object is achieved in the same way as it is done in the engine. TBaseShellEditForm is a super-class to all the different types of edit windows, whereas the TObjectBox is a super-class to the different types of icons on the Performance View. Each type of edit window and icon has a static Create function that is invoked to create it. The TTypeElement class contains a map with function call associated with the type name of the object (see figure 25).

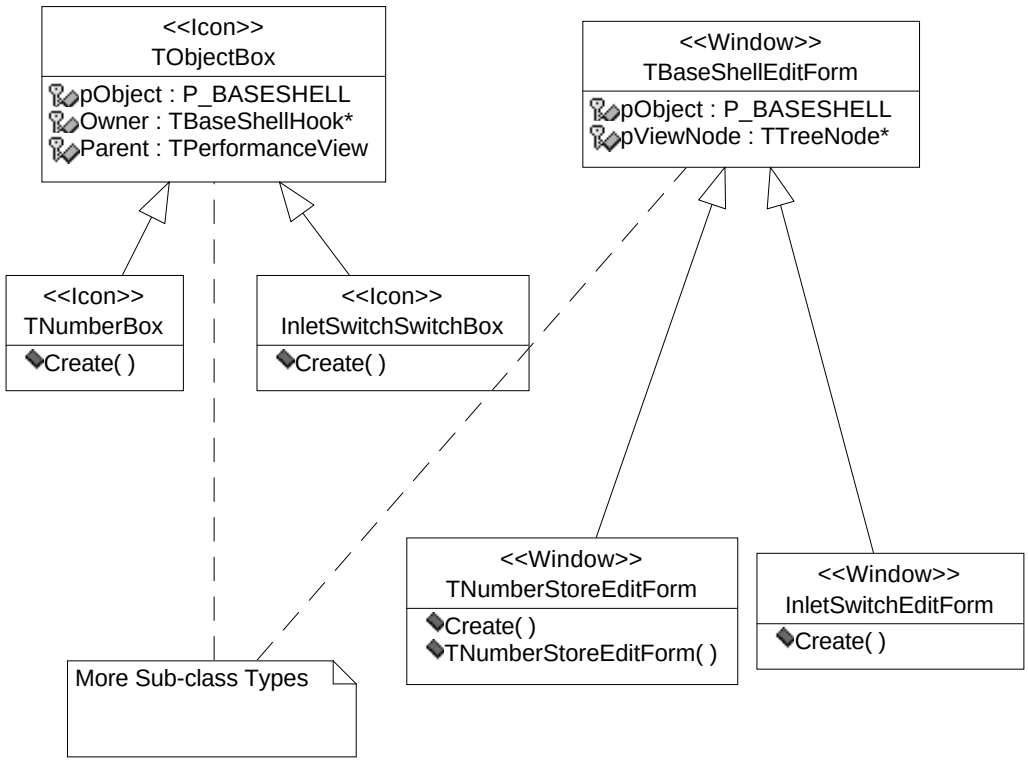

Figure 25 Edit Window and Icon Inheritance Diagram.

DisplayEditForm function call is made to the TBaseShellHook that is associated with the underlying BaseShell. If TBaseShellHook.pEditForm == NULL, a call to TTypeElementList::GetCreateForm(string Type) returns the pointer to the static create function for the edit window associated with the type. The create function is called, which creates a new edit window, and returns a pointer to the new edit window. The pointer is stored inside the TBaseShellHook.pEditForm (see figure 26). TBaseShellHook calls Show upon its pEditForm attribute. The edit window releases all it resources and notifies its owner, which sets pEditForm to NULL, upon closing. The super-class, TBaseShellEditForm has a thread that updates the information on the window when the window is not focused, which enables the user to read and set the underlying values. The sub-classes overload the virtual functions LoadData and StoreButtonClick to enable them to load and store class specific data (see figure 27).

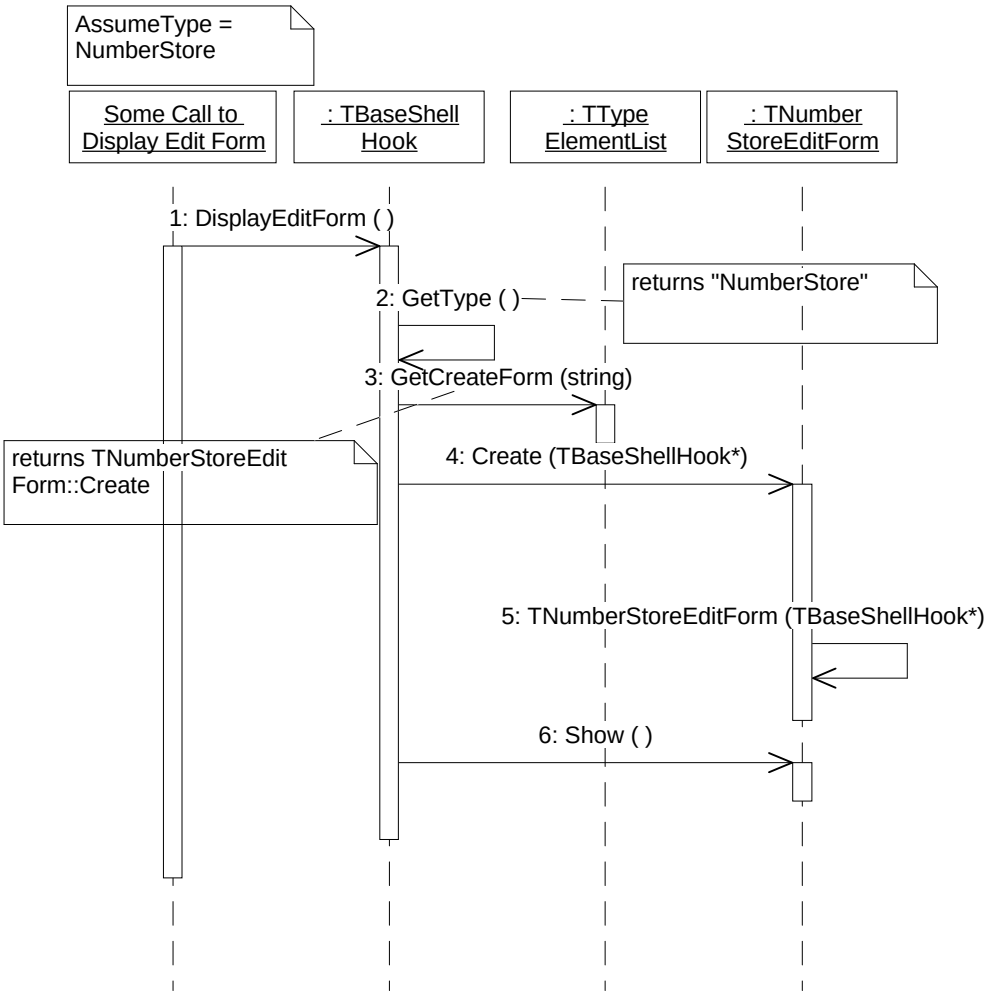

Figure 26 Display Edit Form Sequence.

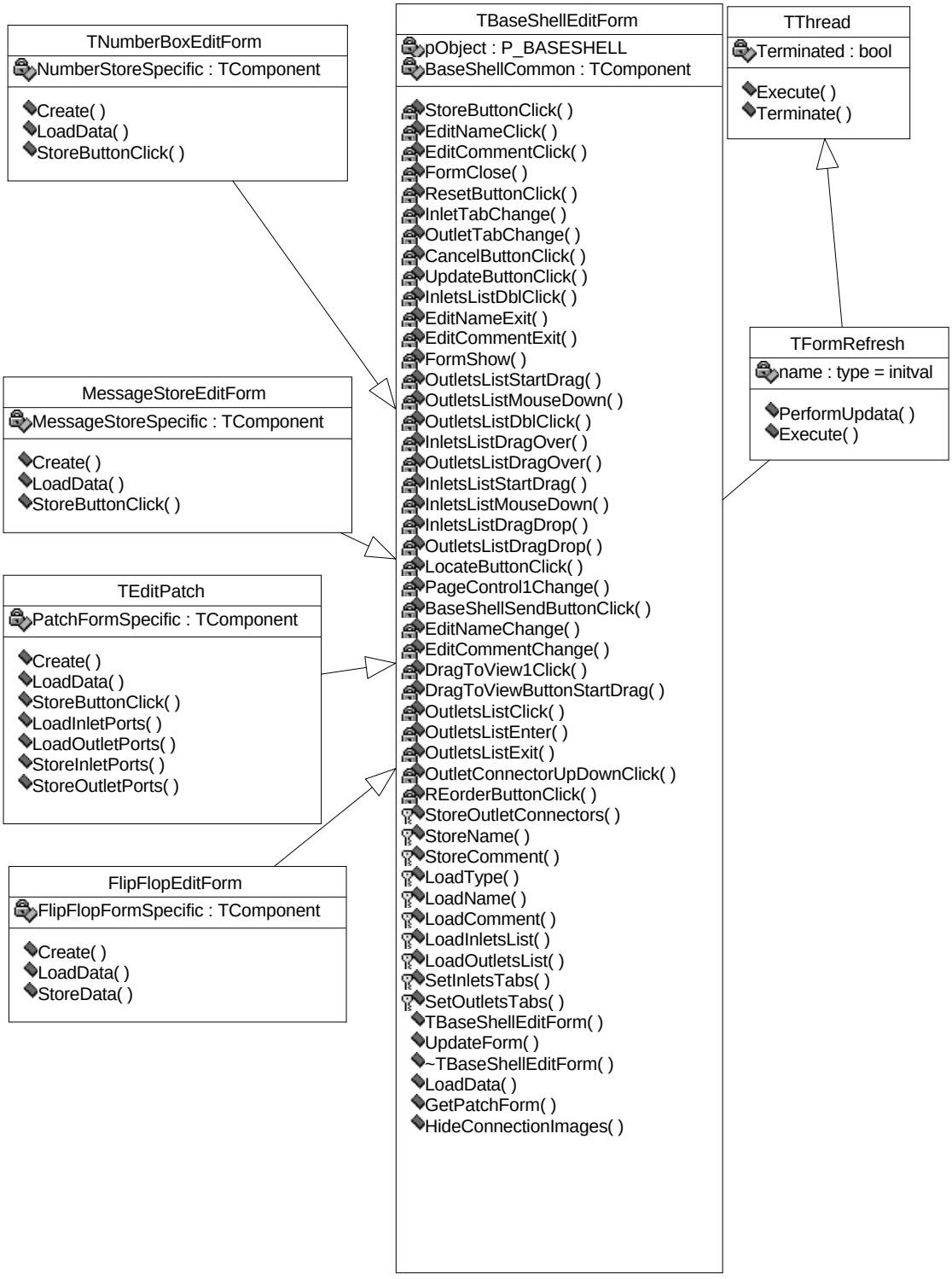

Figure 27 TBaseShellEditForm Class Diagram.

Implementation of the performance view was in two separate packages, both reliant upon the VCL. I created a package specifically for the graphical icons on the

performance view. This will enable me to change the implementation of the icons and lines away from the main program (see figure 28).

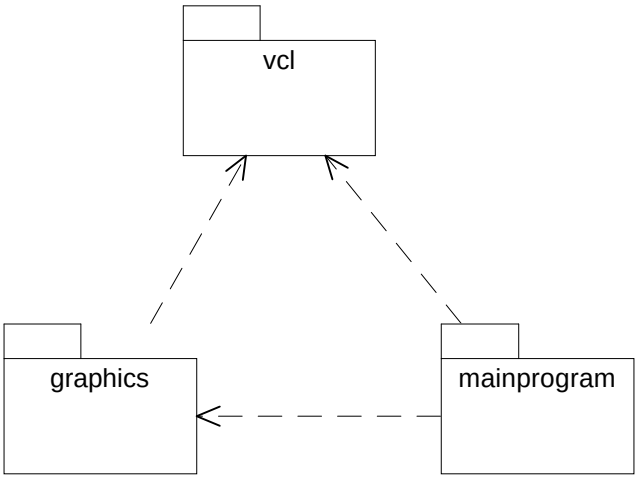

Figure 28 Graphics Separation.

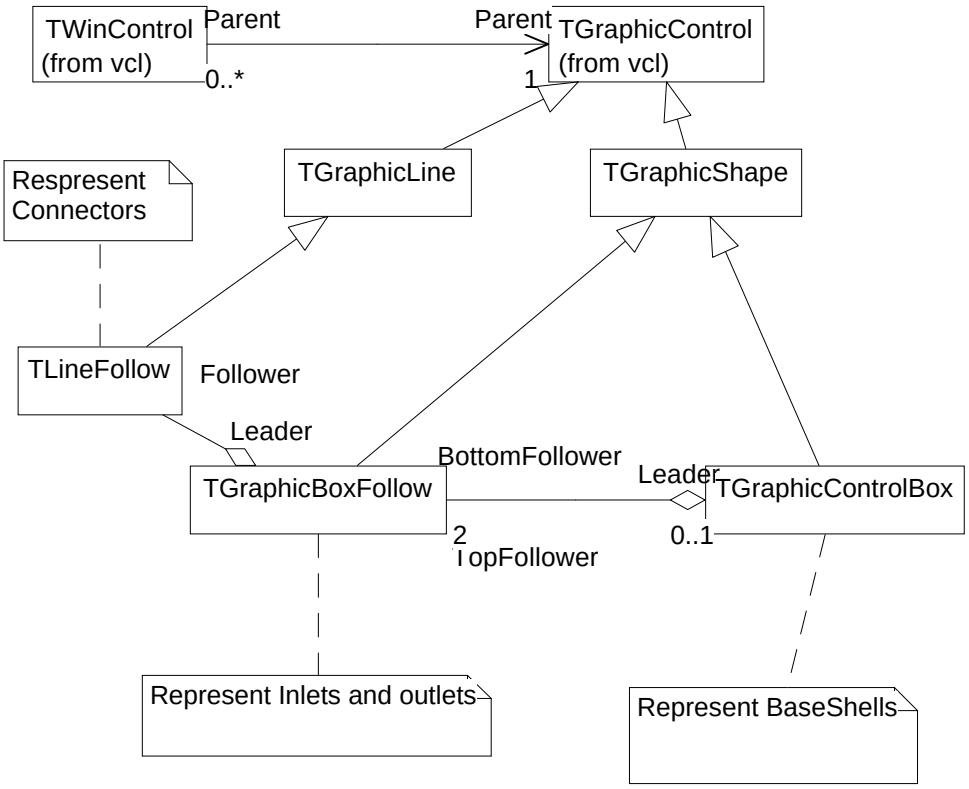

Figure 29 Graphics Inheritance.

TGraphicControl, TGraphicBoxFollow, and TGraphicLine are descended from TGraphicControl in the VCL library, which enables these sub-classes to inherit the ability to respond to the mouse events, and also to display images upon a canvas. TGraphicControl must exist on a TWinControl type object from the VCL (see figure 29). TGraphicControl has two vectors of pointers to TGraphicBoxFollow type objects, which have a vector of pointers to TLineFollow type objects (see figure 30). The leaders notify the followers to follow, which cause the followers to update their position through their leader. TGraphicControl calls the function NotifyFollowers upon itself, which iterates through the TopFollowers and BottomFollowers vectors to call NotifyFollowers in each TGraphicBoxFollow object. TGraphicBoxFollow gets its position from TGraphicControl, and calls the function NotifyFollowers on itself, which iterates through its vector of pointers to TLineFollowers, which update their position (see figure 31)

.

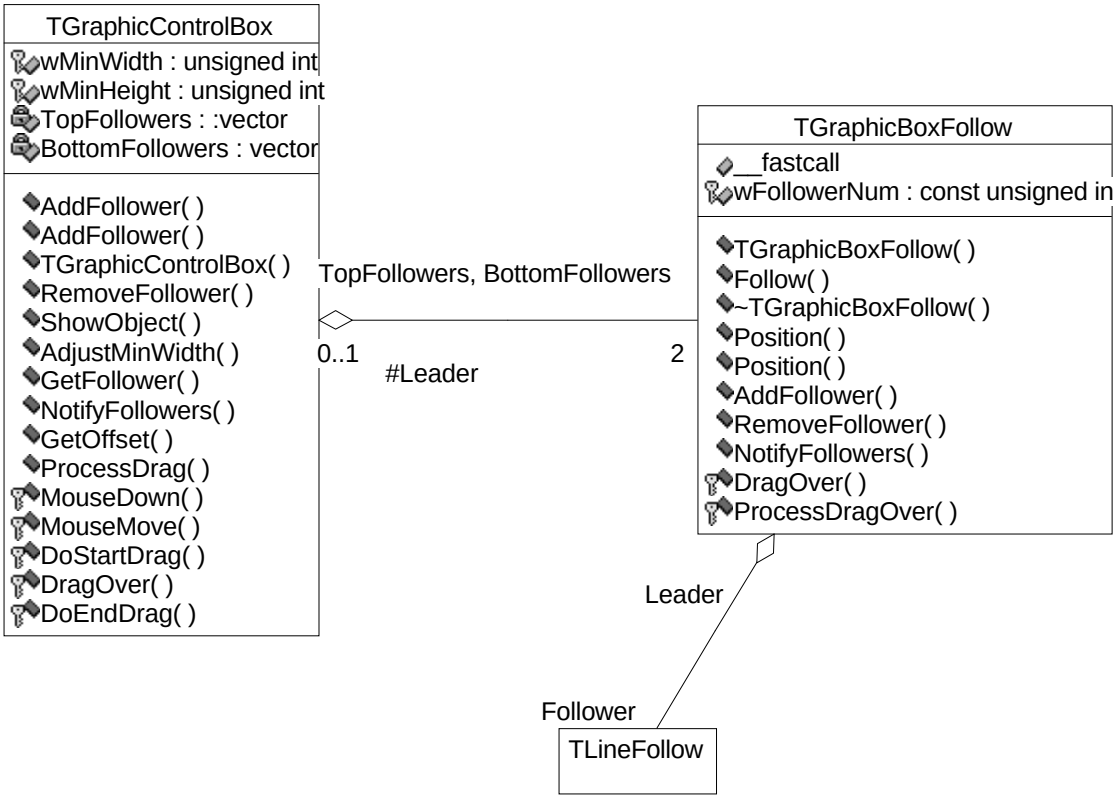

Figure 30 Graphics Icons Relationship Diagram.

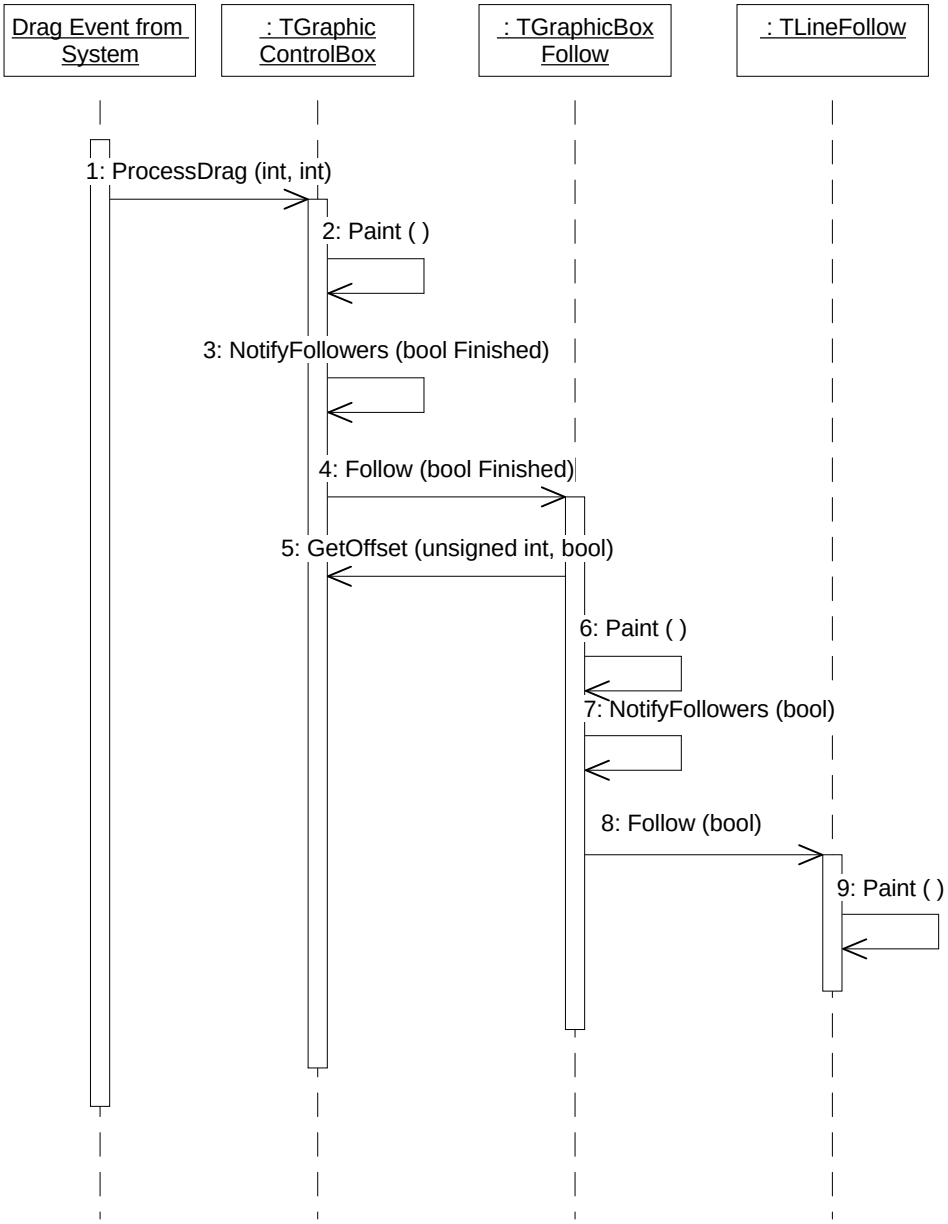

Figure 31 Graphic Package Follow Sequence.

The next level of abstraction associates sub-classes from the Graphic Package to the BaseShell and Connector classes. TObjectBox descends from TGraphicControl and represents the BaseShell. TObjectBox is owned by TBaseShellHook but has TViewForm as its parent. TObjectSocket represents the inlets and outlets of the BaseShell. Its position in the TopFollower or BottomFollower determines its inlet or

outlet number, while the attribute fIsTop determines whether it is an inlet or outlet (see figure 32).

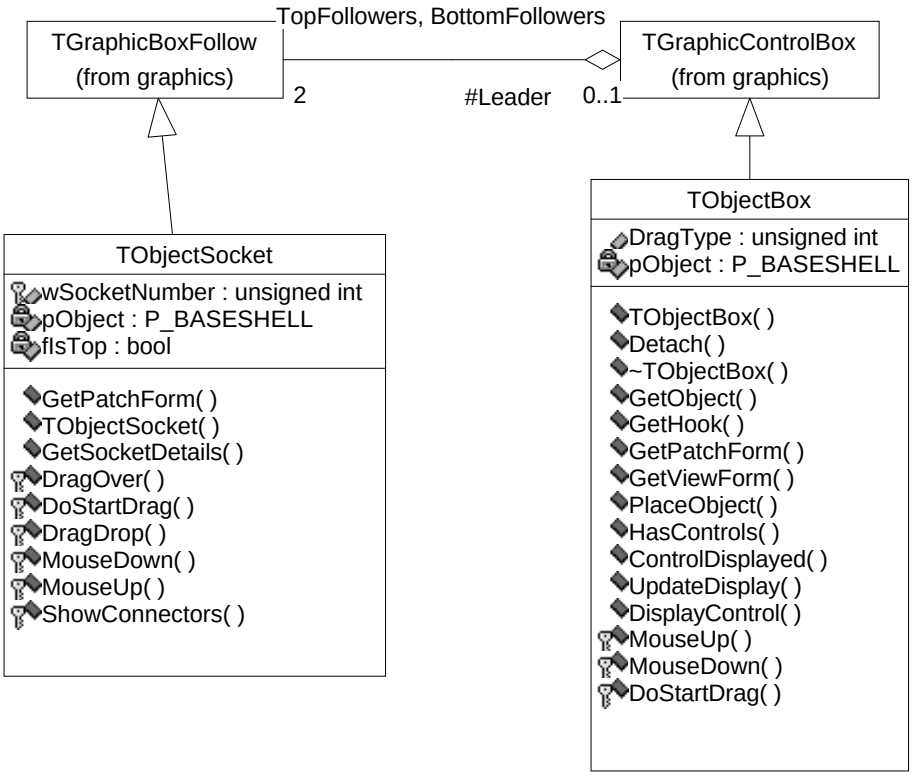

Figure 32 TObjectBox and TObjectSocket.

TObjectBox determines how many inlets or outlets it requires through calls to BaseShellGetNumInlets and BaseShellGetNumOutlets in the bridge, creates and then stores pointers to the subsequent TObjectSocket objects in its TopFollowers and BottomFollowers vectors. The name of the inlet or outlet is acquired through the bridge and stored in the Hint property of the TObjectSocket. The enables the name of the inlet or outlet to display next to the mouse or in the status bar when the mouse moves over the inlet or outlet of the object (see figure 33).

Creating a connection between objects on the performance view is done through a drag and drop operation. The TObjectSocket creates a Connector that is not connected to anything, and a TDragConnector Object.

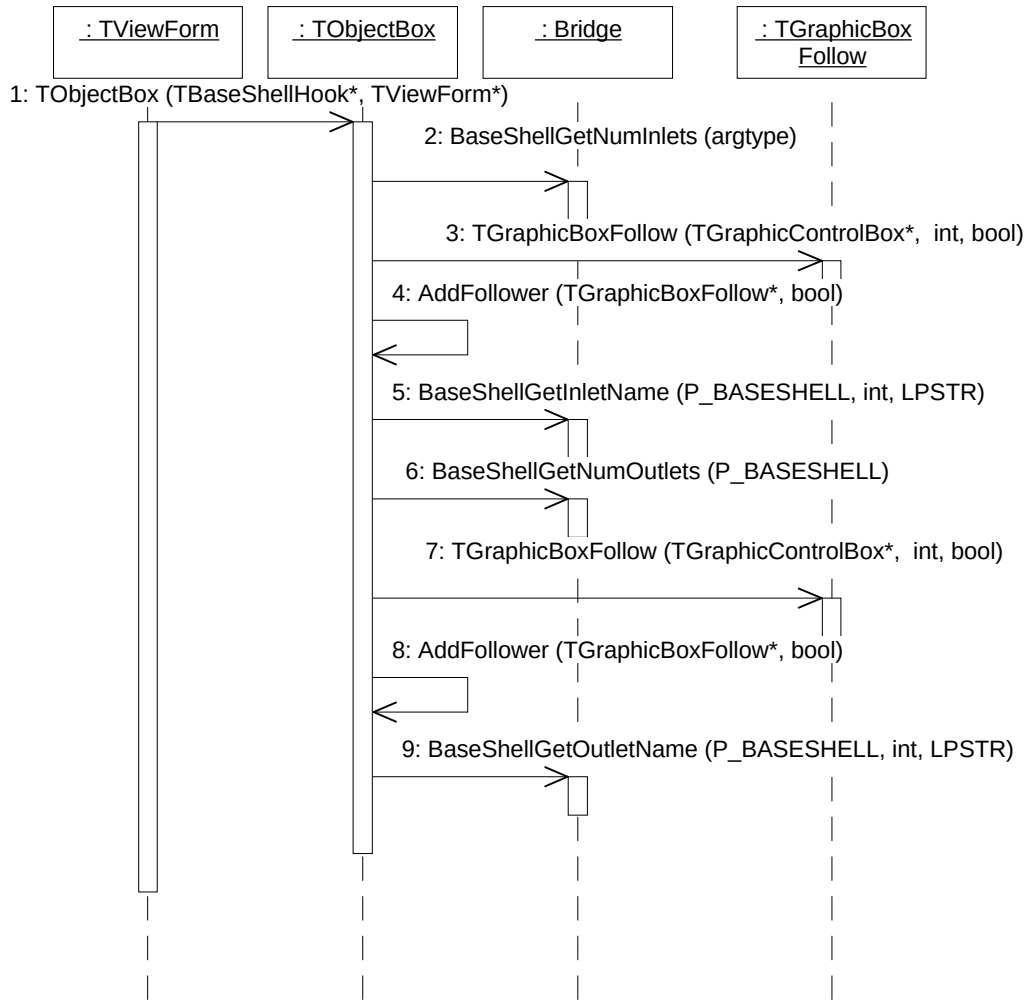

Figure 33 TObjectBox Creation.

Upon starting a drag, which is done by moving the mouse over the socket and holding the left mouse button down, the cursor changes into a hand holding a plug. The inlet or outlet details of the Connector are set or unset by moving mouse over another socket. The TObjectSocket queries the BaseShell through the bridge as to whether it can accept a connection. If a connection is acceptable, the cursor changes into a plug with no hand, otherwise it remains held. Upon drag end, if the TDragConnector was accepted by a TObjectSocket, the TObjectSocket calls Connect to the TDragConnector, which calls BaseShellConnect function through the bridge and adds a new TConnectorHook to the user interface through TPatchForm, otherwise the

Connector is deleted through the TDragConnector. TPatchForm queries all performance views as to whether it can add this particular connector (see figure 34).

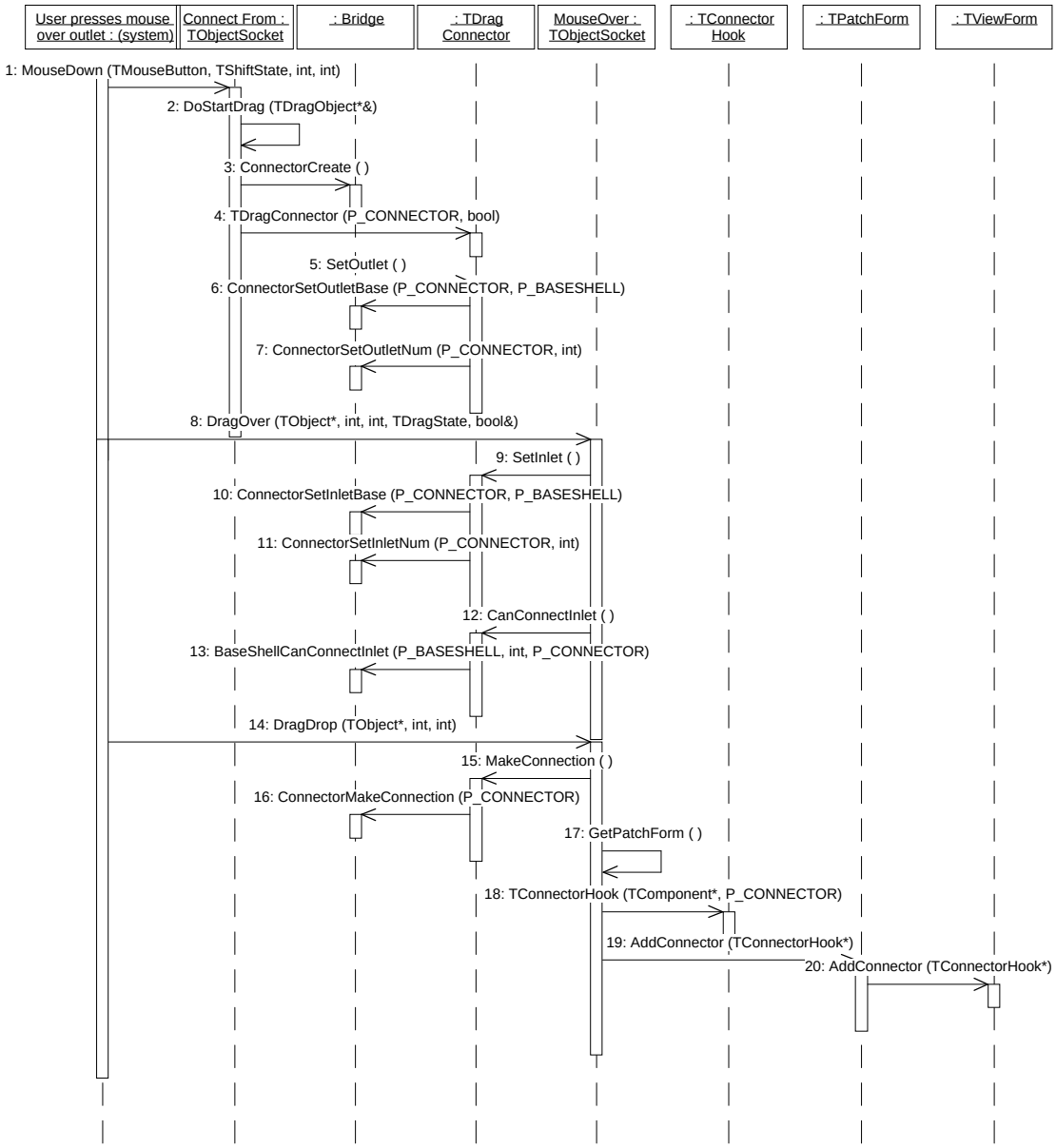

Figure 34 Performance View Make Connection Sequence Diagram.

TViewForm first checks whether it already has the Connector in its TConnectorMap. It then gets the inlet and outlet BaseShell pointers and checks whether they are both in the ObjectMap. If both are in the map, TViewForm gets the TObjectBox associated with the BaseShell pointers and using the inlet and outlet

numbers, gets the TObjectSocket associated with the inlet and outlet number. A new TConnectorLine is created and becomes associated with the TObjectSocket through their super-class functions (see figures 35 and 36).

The patch is stored in two files, one each for the engine and interface. The performance views are stored with details of the object position within the patch instead of the old pointer value as used in the streaming of the engine. Future implementations will use a single file for both parts.

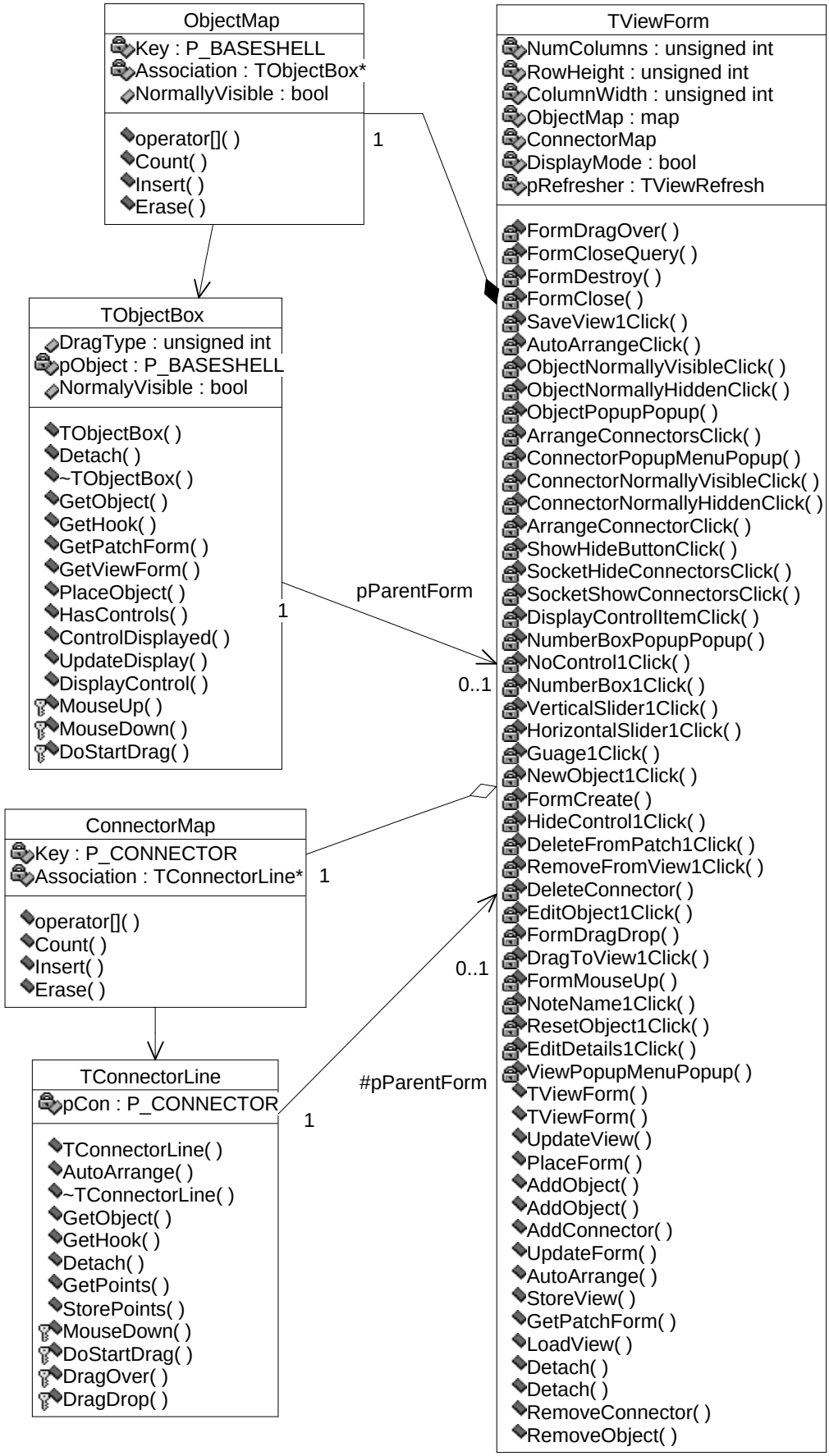

Figure 35 Performance View Class Diagram.

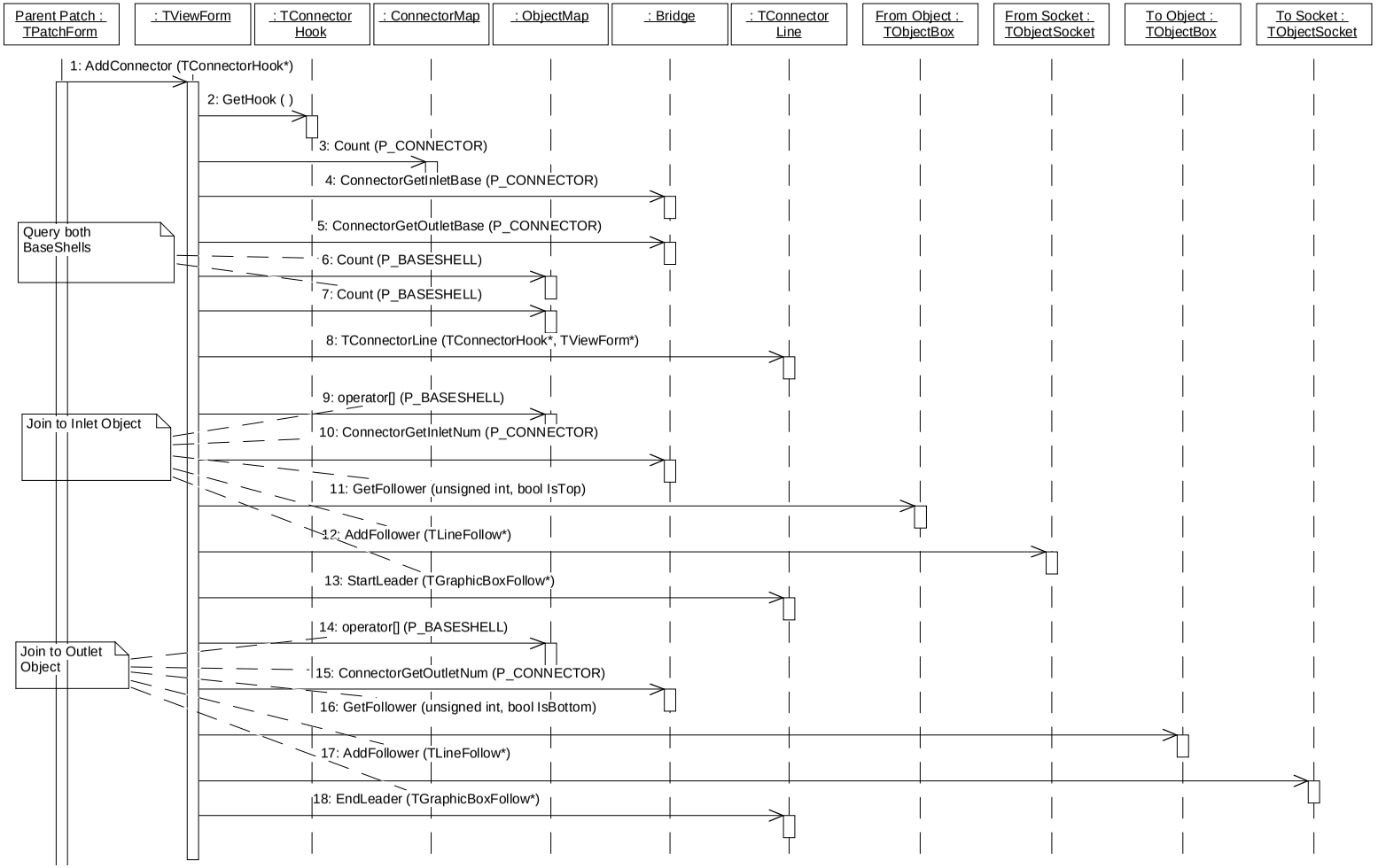

Figure 36 Performance View Add Connector Sequence.

Conclusion.

The current version of Algorithmic Composer has met the specifications I set for myself at the beginning of the project, and has more features than I had originally anticipated. I was able to find limitations to the program while using it to compose. In some instances I made immediate changes to the program, while in other instances I decided to improve later – including the timing algorithm. I have included a compact disk containing three pieces that demonstrate some musical applications of Algorithmic Composer. Each track demonstrates an aspect of the program that fulfilled a musical purpose.

Track one, *Be Still*, is a new work of mine that is still in its compositional stage. I have been able to experiment with different sound textures and timing structures using Algorithmic Composer. The work is recorded live and demonstrates the ability of Algorithmic Composer to enable the composer to experiment with multiple MIDI channels simultaneously.

Track two, *Table of Knowledge*, is written by Jeff Dunn and performed by Jeff Dunn and myself. This was Jeff's first exposure to Algorithmic Composer and demonstrates the ease at which a composer can use this program. I worked through the program operation and patch construction with Jeff for approximately thirty minutes, after which time Jeff was able to construct the patch and compose this piece with virtually no input from me even though he has had no previous experience with MAX.

Track three is an excerpt from *Whirling Wheels*, a work I wrote in 1997 using MAX. I included this excerpt to demonstrate the ability of using Algorithmic Composer for pieces created in MAX. Apart from manipulating sliders on the

computer display with the mouse, I perform *Whirling Wheels* through note, controller, and pitch bend messages from a MIDI keyboard.

I am currently organizing demonstrations of Algorithmic Composer to musicians from the Newcastle Conservatorium of Music through Terry Latham, who is a lecturer in Music Technology there. I plan to continue development of this package for my musical career as both a composer and performer, for the benefit of the University of Western Sydney, and for all musicians who choose to compose and perform with these types of tools.

Bibliography

Dale, Nell and Chip Weems. *Introduction to Pascal and Structured design*, fourth edition. Boston: Jones and Bartlett, 1997.

"Handling Data Types Not Supported by the Thunk Compiler," *Microsoft Software Development Network Online Library*: http://premium.microsoft.com/isapi/devonly/prodinfo/msdnprod/msdnlib.idc?theURL=/msdn/library/sdkdoc/win95/tc_2ugj.htm.

Inprise newsgroups. Server: forums.inprise.com, newsgroup: borland.public.cpp, 20 September 1998. www.inprise.com.

Lippman, Stanley B. and Josée Lajoie. *C++ Primer*, third edition. Reading, Massachusetts: Addison-Wesley, 1988.

McCulley, Mark. "Overcoming Timer-Latency Problems in MIDI Sequencers," *Microsoft Software Development Network Online Library* (1996): http://premium.microsoft.com/isapi/devonly/prodinfo/msdnprod/msdnlib.idc?theURL=/msdn_mlatency.htm.

Messick, Paul. *Maximum MIDI: Music Applications in C++*. Greenwich CT: Manning, 1988.

Petzold, Charles. *Programming Windows 95*, fourth edition. Redmond: Microsoft Press, 1996.

Pressing, Jeff. *Synthesizer Performance and Real-Time Technique. Oxford*: Oxford University Press, 1992.

Reisdorph, Kent. *Teach Yourself Borland C++ Builder in 14Days*. Indiana: SAMS, 1998.

Header Files Required for Interfacing to Engine

```
/*****************************************************************************************
*    TypeNames.h
*    this contains definitions of the type classes and the type names associated with them
*****************************************************************************************/

#ifndef TYPENAMES_H
#define TYPENAMES_H

#define INLETPORT_TYPE  "PatchInletPort"
#define OUTLETPORT_TYPE  "PatchOutletPort"
#define CONNECTOR_TYPE "Connector"
#define CALCULATE_TYPE "Calculate"
#define COUNTER_TYPE  "Counter"
#define DELAY_TYPE  "Delay"
#define DISPLAY_TYPE "Display"
#define FLIPFLOP_TYPE  "FlipFlop"
#define MESSAGESTORE_TYPE  "MessageStore"
#define METRO_TYPE  "Metro"
#define PATCH_TYPE  "Patch"
#define SELECTOR_TYPE "Selector"
#define STDMIDIOUT_TYPE "Midi Out"
#define STDMIDIIN_TYPE "Midi In"
#define SWITCH_OUTLETS_TYPE  "Outlets Switch"
#define SWITCH_INLETS_TYPE  "Inlets Switch"
#define TOGGLE_TYPE  "Toggle"
#define TRIGGER_TYPE "Trigger"
#define PATCH_FILE_ID "Patch from File"
#define TABLE_TYPE "Table"
```

```
#define NUMBERSTORE_TYPE "Number Store"
#define RANDOMGEN_TYPE "Random Gen"
#define SEQUENCER_TYPE "Sequencer"
#endif


/*****************************************************************************************************
*    WindowsMessages.h
*    Defines the window message for calls using PostMessage
*****************************************************************************************************/

#ifndef WINDOWS_MESSAGES_H
#define WINDOWS_MESSAGES_H
//define windows messages here

#define DISPLAY_DELETED (WM_USER +1)
#define DISPLAY_MESSAGE (WM_USER + 2)
#define ERROR_MESSAGE   (WM_USER + 3)
#define TRIGGER_MESSAGE (WM_USER + 4)
#define TRIGGER_DELETED (WM_USER + 5)

#endif
```

```
/*****************************************************************************************
*   Bridge.h
*   Header for the Algorithmic Composer Engine Bridge
*   Declares all functions required to interface to the engine
*
*****************************************************************************************/
#ifndef BRIDGE_H
#define BRIDGE_H
#include "Typedefs.h"

#ifdef _PASS_CLASS_POINTERS //only for debugging with the engine
#include "Patch.h"
#endif
#ifdef __cplusplus
extern "C" {          // Assume C declarations for C++
#endif

/*************************************************************************
Common Interface functions
*************************************************************************/
// Conversion of MIDI
PREFIX void WINAPI EXPORT NumberToMidi(unsigned NumVal, LPSTR Buf);
PREFIX bool WINAPI EXPORT MidiToNumber(LPSTR szNoteName, int* Result);

//Error Interface
PREFIX    bool WINAPI EXPORT GetErrorMessage(LPSTR Buf);
PREFIX    void WINAPI EXPORT SetErrorWindow(HWND hWnd);

//all remaining functions assume the first argument is a valid
//if it is a pointer represenatation
```

```
/*****************************************************************
Identity class access functions
*****************************************************************/

PREFIX    void WINAPI EXPORT IdentityGetName(P_IDENTITY dwpIdentity, LPSTR Buf);
PREFIX    void WINAPI EXPORT IdentityGetComment(P_IDENTITY dwpIdentity, LPSTR Buf);
PREFIX    void WINAPI EXPORT IdentityGetType(P_IDENTITY dwpIdentity, LPSTR Buf);

PREFIX    void WINAPI EXPORT IdentitySetName(P_IDENTITY dwpIdentity, LPSTR Buf);
PREFIX    void WINAPI EXPORT IdentitySetComment(P_IDENTITY dwpIdentity, LPSTR Buf);
PREFIX    void WINAPI EXPORT IdentitySetModified(P_IDENTITY dwpIdentity, bool Modified);
PREFIX    bool WINAPI EXPORT IdentityGetModified(P_IDENTITY dwpIdentity);
/*****************************************************************
BaseShell class access functions
*****************************************************************/

PREFIX    P_BASESHELL WINAPI EXPORT BaseShellCreate(LPSTR Type, P_PATCH pParent);

PREFIX    P_PATCH WINAPI EXPORT BaseShellGetParent(P_BASESHELL dwpShell);
PREFIX    void WINAPI EXPORT BaseShellSetParent(P_BASESHELL dwpShell, P_PATCH dwpParent);
PREFIX    void WINAPI EXPORT BaseShellReset(P_BASESHELL dwpShell);
PREFIX    void WINAPI EXPORT BaseShellSend(P_BASESHELL dwpShell);
PREFIX    bool WINAPI EXPORT BaseShellCanSave(P_BASESHELL dwpShell);

//get size
//call before using the remaining functions
PREFIX    unsigned WINAPI EXPORT BaseShellGetNumInlets(P_BASESHELL dwpShell);
PREFIX    unsigned WINAPI EXPORT BaseShellGetNumOutlets(P_BASESHELL dwpShell);

//get name for hints over inlet or outlet
```

```
//all following assume wSocketNumber in range
PREFIX    void WINAPI EXPORT BaseShellGetInletName(P_BASESHELL dwpShell, unsigned wSocketNum, LPSTR Buf);
PREFIX    void WINAPI EXPORT BaseShellGetOutletName(P_BASESHELL dwpShell, unsigned wSocketNum, LPSTR Buf);

//how many connectors for a particular inlet or outlet
PREFIX    unsigned WINAPI EXPORT BaseShellGetNumInletConnectors(P_BASESHELL dwpShell,unsigned wSocketNum);
PREFIX    unsigned WINAPI EXPORT BaseShellGetNumOutletConnectors(P_BASESHELL dwpShell, unsigned wSocketNum);

//test for ability to connect
PREFIX    bool WINAPI EXPORT BaseShellCanConnectInlet(P_BASESHELL dwpShell, unsigned wSocketNum, P_CONNECTOR dwpCon);
PREFIX    bool WINAPI EXPORT BaseShellCanConnectOutlet(P_BASESHELL dwpShell, unsigned wSocketNum, P_CONNECTOR
dwpCon);

//get connector at index
//assumes Index is valid
PREFIX    P_CONNECTOR WINAPI EXPORT BaseShellGetInletConnector(P_BASESHELL dwpShell,unsigned wSocketNum, unsigned
Index);
PREFIX    P_CONNECTOR WINAPI EXPORT BaseShellGetOutletConnector(P_BASESHELL dwpShell, unsigned wSocketNum, unsigned
Index);

// find the Connector
// assumes dwpCon is in this socket
PREFIX    unsigned WINAPI EXPORT BaseShellFindOutletConnectorIndex(P_BASESHELL dwpShell, unsigned wSocketNum,
P_CONNECTOR dwpCon);
PREFIX    unsigned WINAPI EXPORT BaseShellFindInletConnectorIndex(P_BASESHELL dwpShell, unsigned wSocketNum,
P_CONNECTOR dwpCon);

//assumes both indexes are valid
PREFIX    void WINAPI EXPORT BaseShellSwapOutletConnector(P_BASESHELL dwpShell, unsigned OutletNumber, unsigned Connector1,
unsigned Connector2);
```

```
/*****************************************************************
Connector class access functions
*****************************************************************/
//test if full connection exists
PREFIX    bool WINAPI EXPORT ConnectorIsValid(P_CONNECTOR dwpCon);

//find out about Connector
PREFIX    P_BASESHELL WINAPI EXPORT ConnectorGetInletBase(P_CONNECTOR dwpCon);
PREFIX    unsigned WINAPI EXPORT ConnectorGetInletNum(P_CONNECTOR dwpCon);
PREFIX    P_BASESHELL WINAPI EXPORT ConnectorGetOutletBase(P_CONNECTOR dwpCon);
PREFIX    unsigned WINAPI EXPORT ConnectorGetOutletNum(P_CONNECTOR dwpCon);

//find position in Outlet
PREFIX    unsigned WINAPI EXPORT ConnectorGetOutletIndex(P_CONNECTOR dwpCon);

PREFIX    P_PATCH WINAPI EXPORT ConnectorGetParent(P_CONNECTOR dwpCon);

//use for testing acceptability of connector
//does not actually connect to the BaseShell
PREFIX    void WINAPI EXPORT ConnectorSetInletBase(P_CONNECTOR dwpCon, P_BASESHELL dwpShell);
PREFIX    void WINAPI EXPORT ConnectorSetInletNum(P_CONNECTOR dwpCon, unsigned wSocketNum);
PREFIX    void WINAPI EXPORT ConnectorSetOutletBase(P_CONNECTOR dwpCon, P_BASESHELL dwpShell);
PREFIX    void WINAPI EXPORT ConnectorSetOutletNum(P_CONNECTOR dwpCon, unsigned wSocketNum);

//complete connection.
PREFIX    bool WINAPI EXPORT ConnectorMakeConnection(P_CONNECTOR dwpCon);
PREFIX    void WINAPI EXPORT ConnectorSetParent(P_CONNECTOR dwpCon, P_PATCH dwpParent);

PREFIX    P_CONNECTOR WINAPI EXPORT ConnectorCreate();
```

```
PREFIX    void WINAPI EXPORT ConnectorDelete(P_CONNECTOR dwpCon);

PREFIX    bool WINAPI EXPORT ConnectorCanSave(P_CONNECTOR dwpCon);

/*****************************************************************************
*   Patch Class access functions
*****************************************************************************/
//assumes dwpShell exists in this patch
PREFIX    void WINAPI EXPORT PatchDetachBaseShell(P_PATCH dwpPatch, P_BASESHELL dwpShell);
PREFIX    unsigned WINAPI EXPORT PatchFindBaseShellPosition(P_PATCH dwpPatch, P_BASESHELL dwpShell);

PREFIX    void WINAPI EXPORT PatchAddBaseShell(P_PATCH dwpPatch, P_BASESHELL dwpShell);

PREFIX    unsigned WINAPI EXPORT PatchNumberBaseShells(P_PATCH dwpPatch);
//assumes a valid index
PREFIX    P_BASESHELL WINAPI EXPORT PatchFindBaseShell(P_PATCH dwpPatch, unsigned Index);

//causes Connector to become deleted from memory
PREFIX    void WINAPI EXPORT PatchDetachConnector(P_PATCH dwpPatch, P_CONNECTOR dwpCon);

//call after completeing a connection
PREFIX    void WINAPI EXPORT PatchAddConnector(P_PATCH dwpPatch, P_CONNECTOR dwpCon);

PREFIX    unsigned WINAPI EXPORT PatchNumberConnectors(P_PATCH dwpPatch);
//assumes a valid index
PREFIX    P_CONNECTOR WINAPI EXPORT PatchFindConnector(P_PATCH dwpPatch, unsigned Index);

//call after deleting any BaseShells to see if any Connectors were also deleted
PREFIX    P_CONNECTOR WINAPI EXPORT PatchGetDeadConnector(P_PATCH dwpPatch);
```

```
//same as BaseShellCreate for Patch Inlets and Outlets
PREFIX    P_BASESHELL WINAPI EXPORT PatchAddInlet(P_PATCH dwpPatch);
PREFIX    P_BASESHELL WINAPI EXPORT PatchAddOutlet(P_PATCH dwpPatch);
PREFIX    P_PATCH WINAPI EXPORT PatchMakeNew(LPSTR Name);

//streaming
PREFIX    void WINAPI EXPORT PatchSetFileName(P_PATCH dwpPatch, LPSTR FileName);
PREFIX    P_PATCH WINAPI EXPORT PatchLoadPatchFile(LPSTR FileName, P_PATCH dwpParent);
PREFIX    void WINAPI EXPORT PatchGetFileName(P_PATCH dwpPatch, LPSTR FileName);
PREFIX    bool WINAPI EXPORT PatchSave(P_PATCH dwpPatch, LPSTR FileName);
PREFIX    void WINAPI EXPORT PatchDelete(P_PATCH dwpPatch);
PREFIX    void WINAPI EXPORT PatchSetNumViews(P_PATCH dwpPatch, unsigned NumViews);
PREFIX    unsigned WINAPI EXPORT PatchGetNumViews(P_PATCH dwpPatch);

//alter the ordering of inlets and outlets
PREFIX    void WINAPI EXPORT PatchSwapOutlets(P_PATCH dwpPatch, unsigned Outlet1, unsigned Outlet2);
PREFIX    void WINAPI EXPORT PatchSwapInlets(P_PATCH dwpPatch, unsigned Inlet1, unsigned Inlet2);

PREFIX    P_BASESHELL WINAPI EXPORT PatchGetInlet(P_PATCH dwpPatch, unsigned SocketNum);
PREFIX    P_BASESHELL WINAPI EXPORT PatchGetOutlet(P_PATCH dwpPatch, unsigned SocketNum);

PREFIX    unsigned WINAPI EXPORT PatchOutletGetOutletNumber(P_BASESHELL dwpPort);
PREFIX    unsigned WINAPI EXPORT PatchInletGetInletNumber(P_BASESHELL dwpPort);

/*****************************************************************************
*   Calculate Class access functions
*****************************************************************************/
PREFIX int WINAPI EXPORT CalculateGetOperator(P_BASESHELL dwpCalc);
PREFIX int WINAPI EXPORT CalculateGetResetOperator(P_BASESHELL dwpCalc);
PREFIX int WINAPI EXPORT CalculateGetLValue(P_BASESHELL dwpCalc);
```

```
PREFIX int WINAPI EXPORT CalculateGetResetLValue(P_BASESHELL dwpCalc);
PREFIX int WINAPI EXPORT CalculateGetRValue(P_BASESHELL dwpCalc);
PREFIX int WINAPI EXPORT CalculateGetResetRValue(P_BASESHELL dwpCalc);
PREFIX bool WINAPI EXPORT CalculateGetRTriggerCalc(P_BASESHELL dwpCalc);
PREFIX unsigned WINAPI EXPORT CalculateGetNumOperators(void);
PREFIX void WINAPI EXPORT CalculateGetOperatorType(unsigned Index, LPSTR Buf);


PREFIX void WINAPI EXPORT CalculateSetOperator(P_BASESHELL dwpCalc, int Value);
PREFIX void WINAPI EXPORT CalculateSetResetOperator(P_BASESHELL dwpCalc, int Value);
PREFIX void WINAPI EXPORT CalculateSetLValue(P_BASESHELL dwpCalc, int Value);
PREFIX void WINAPI EXPORT CalculateSetResetLValue(P_BASESHELL dwpCalc, int Value);
PREFIX void WINAPI EXPORT CalculateSetRValue(P_BASESHELL dwpCalc, int Value);
PREFIX void WINAPI EXPORT CalculateSetResetRValue(P_BASESHELL dwpCalc, int Value);
PREFIX void WINAPI EXPORT CalculateSetRTriggerCalc(P_BASESHELL dwpCalc, bool Value);

PREFIX void WINAPI EXPORT CalculateCalculateValues(P_BASESHELL dwpCalc);

/*****************************************************************************
*   Counter Class access functions
*****************************************************************************/
PREFIX int WINAPI EXPORT CounterGetCount(P_BASESHELL dwpCount);
PREFIX int WINAPI EXPORT CounterGetResetCount(P_BASESHELL dwpCount);
PREFIX int WINAPI EXPORT CounterGetStepSize(P_BASESHELL dwpCount);
PREFIX int WINAPI EXPORT CounterGetResetStepSize(P_BASESHELL dwpCount);
PREFIX int WINAPI EXPORT CounterGetUpperLimit(P_BASESHELL dwpCount);
PREFIX int WINAPI EXPORT CounterGetResetUpperLimit(P_BASESHELL dwpCount);
PREFIX int WINAPI EXPORT CounterGetLowerLimit(P_BASESHELL dwpCount);
PREFIX int WINAPI EXPORT CounterGetResetLowerLimit(P_BASESHELL dwpCount);
PREFIX bool WINAPI EXPORT CounterGetDirection(P_BASESHELL dwpCount);
```

```
PREFIX bool WINAPI EXPORT CounterGetResetDirection(P_BASESHELL dwpCount);
PREFIX bool WINAPI EXPORT CounterGetBiDirectional(P_BASESHELL dwpCount);
PREFIX bool WINAPI EXPORT CounterGetResetBiDirectional(P_BASESHELL dwpCount);

PREFIX void WINAPI EXPORT CounterSetCount(P_BASESHELL dwpCount, int Value);
PREFIX void WINAPI EXPORT CounterSetResetCount(P_BASESHELL dwpCount, int Value);
PREFIX void WINAPI EXPORT CounterSetStepSize(P_BASESHELL dwpCount, int Value);
PREFIX void WINAPI EXPORT CounterSetResetStepSize(P_BASESHELL dwpCount, int Value);
PREFIX void WINAPI EXPORT CounterSetUpperLimit(P_BASESHELL dwpCount, int Value);
PREFIX void WINAPI EXPORT CounterSetResetUpperLimit(P_BASESHELL dwpCount, int Value);
PREFIX void WINAPI EXPORT CounterSetLowerLimit(P_BASESHELL dwpCount, int Value);
PREFIX void WINAPI EXPORT CounterSetResetLowerLimit(P_BASESHELL dwpCount, int Value);
PREFIX void WINAPI EXPORT CounterSetDirection(P_BASESHELL dwpCount, bool Value);
PREFIX void WINAPI EXPORT CounterSetResetDirection(P_BASESHELL dwpCount, bool Value);
PREFIX void WINAPI EXPORT CounterSetBiDirectional(P_BASESHELL dwpCount, bool Value);
PREFIX void WINAPI EXPORT CounterSetResetBiDirectional(P_BASESHELL dwpCount, bool Value);
/*****************************************************************************
*   Metro Class access functions
*****************************************************************************/
PREFIX unsigned WINAPI EXPORT MetroGetInterval(P_BASESHELL dwpMetro);
PREFIX unsigned WINAPI EXPORT MetroGetResetInterval(P_BASESHELL dwpMetro);
PREFIX bool WINAPI EXPORT MetroIsRunning(P_BASESHELL dwpMetro);

PREFIX void WINAPI EXPORT MetroActivate(P_BASESHELL dwpMetro, bool Go);
PREFIX void WINAPI EXPORT MetroSetInterval(P_BASESHELL dwpMetro, unsigned Value);
PREFIX void WINAPI EXPORT MetroSetResetInterval(P_BASESHELL dwpMetro, unsigned Value);

/*****************************************************************************
*   Delay Class access functions
*****************************************************************************/
```

```
PREFIX unsigned WINAPI EXPORT DelayGetInterval(P_BASESHELL dwpDelay);
PREFIX unsigned WINAPI EXPORT DelayGetResetInterval(P_BASESHELL dwpDelay);
PREFIX void WINAPI EXPORT DelaySetInterval(P_BASESHELL dwpDelay, unsigned Value);
PREFIX void WINAPI EXPORT DelaySetResetInterval(P_BASESHELL dwpDelay, unsigned Value);

/*****************************************************************************
*   Switch Class access functions
*****************************************************************************/
PREFIX unsigned WINAPI EXPORT SwitchGetPosition(P_BASESHELL dwpSwitch);
PREFIX unsigned WINAPI EXPORT SwitchGetResetPosition(P_BASESHELL dwpSwitch);
PREFIX unsigned WINAPI EXPORT SwitchGetNumContacts(P_BASESHELL dwpSwitch);
PREFIX void WINAPI EXPORT SwitchSetPosition(P_BASESHELL dwpSwitch, unsigned Value);
PREFIX void WINAPI EXPORT SwitchSetResetPosition(P_BASESHELL dwpSwitch, unsigned Value);
PREFIX void WINAPI EXPORT SwitchResize(P_BASESHELL dwpSwitch, unsigned Value);

/*****************************************************************************
*   StdMidiOut Class access functions
*****************************************************************************/
PREFIX unsigned WINAPI EXPORT StdMidiOutGetStatus(P_BASESHELL dwpMidi);
PREFIX unsigned WINAPI EXPORT StdMidiOutGetResetStatus(P_BASESHELL dwpMidi);
PREFIX unsigned WINAPI EXPORT StdMidiOutGetChannel(P_BASESHELL dwpMidi);
PREFIX unsigned WINAPI EXPORT StdMidiOutGetResetChannel(P_BASESHELL dwpMidi);
PREFIX unsigned WINAPI EXPORT StdMidiOutGetData1(P_BASESHELL dwpMidi);
PREFIX unsigned WINAPI EXPORT StdMidiOutGetResetData1(P_BASESHELL dwpMidi);
PREFIX unsigned WINAPI EXPORT StdMidiOutGetData2(P_BASESHELL dwpMidi);
PREFIX unsigned WINAPI EXPORT StdMidiOutGetResetData2(P_BASESHELL dwpMidi);
PREFIX bool WINAPI EXPORT StdMidiOutGetMono(P_BASESHELL dwpMidi);
PREFIX bool WINAPI EXPORT StdMidiOutGetResetMono(P_BASESHELL dwpMidi);
PREFIX bool WINAPI EXPORT StdMidiOutGetData1Trigger(P_BASESHELL dwpMidi);
PREFIX bool WINAPI EXPORT StdMidiOutGetResetData1Trigger(P_BASESHELL dwpMidi);
```

```
PREFIX unsigned WINAPI EXPORT StdMidiOutGetDevice(P_BASESHELL dwpMidi);
PREFIX bool WINAPI EXPORT StdMidiOutSetStatus(P_BASESHELL dwpMidi, unsigned Value);
PREFIX bool WINAPI EXPORT StdMidiOutSetResetStatus(P_BASESHELL dwpMidi, unsigned Value);
PREFIX bool WINAPI EXPORT StdMidiOutSetChannel(P_BASESHELL dwpMidi, unsigned Value);
PREFIX bool WINAPI EXPORT StdMidiOutSetResetChannel(P_BASESHELL dwpMidi, unsigned Value);
PREFIX bool WINAPI EXPORT StdMidiOutSetData1(P_BASESHELL dwpMidi, unsigned Value);
PREFIX bool WINAPI EXPORT StdMidiOutSetResetData1(P_BASESHELL dwpMidi, unsigned Value);
PREFIX bool WINAPI EXPORT StdMidiOutSetData2(P_BASESHELL dwpMidi, unsigned Value);
PREFIX bool WINAPI EXPORT StdMidiOutSetResetData2(P_BASESHELL dwpMidi, unsigned Value);
PREFIX void WINAPI EXPORT StdMidiOutSetData1Trigger(P_BASESHELL dwpMidi, bool Value);
PREFIX void WINAPI EXPORT StdMidiOutSetResetData1Trigger(P_BASESHELL dwpMidi, bool Value);
PREFIX void WINAPI EXPORT StdMidiOutSetMono(P_BASESHELL dwpMidi, bool Value);
PREFIX void WINAPI EXPORT StdMidiOutSetResetMono(P_BASESHELL dwpMidi, bool Value);
PREFIX void WINAPI EXPORT StdMidiOutSetDevice(P_BASESHELL dwpMidi, unsigned Value);

PREFIX void WINAPI EXPORT StdMidiOutSend(P_BASESHELL dwpMidi);
PREFIX void WINAPI EXPORT StdMidiOutFlushDevice(P_BASESHELL dwpMidi);
PREFIX void WINAPI EXPORT StdMidiOutResetDevice(P_BASESHELL dwpMidi);

PREFIX void WINAPI EXPORT StdMidiOutAddDevice(DWORD Handle);
PREFIX void WINAPI EXPORT StdMidiOutClearDevices(void);
/*****************************************************************************
Midi In Functions
*****************************************************************************/
//these are the filters
PREFIX unsigned WINAPI EXPORT StdMidiInGetStatus(P_BASESHELL dwpMidi);
PREFIX unsigned WINAPI EXPORT StdMidiInGetChannel(P_BASESHELL dwpMidi);
PREFIX int WINAPI EXPORT StdMidiInGetData1(P_BASESHELL dwpMidi);
PREFIX int WINAPI EXPORT StdMidiInGetData2(P_BASESHELL dwpMidi);
PREFIX unsigned WINAPI EXPORT StdMidiInGetDevice(P_BASESHELL dwpMidi);
```

```
PREFIX bool WINAPI EXPORT StdMidiInGetFiltered(P_BASESHELL dwpMidi);
PREFIX bool WINAPI EXPORT StdMidiInGetNoteOnZeroFilter(P_BASESHELL dwpMidi);
PREFIX bool WINAPI EXPORT StdMidiInSetStatus(P_BASESHELL dwpMidi, unsigned Value);
PREFIX bool WINAPI EXPORT StdMidiInSetChannel(P_BASESHELL dwpMidi, unsigned Value);
PREFIX void WINAPI EXPORT StdMidiInSetDevice(P_BASESHELL dwpMidi, unsigned Value);
PREFIX bool WINAPI EXPORT StdMidiInSetData1(P_BASESHELL dwpMidi, int Value);
PREFIX bool WINAPI EXPORT StdMidiInSetData2(P_BASESHELL dwpMidi, int Value);
PREFIX void WINAPI EXPORT StdMidiInProcessMidiIn(unsigned wDevice, LPMIDIEVENT lpEvent);
PREFIX void WINAPI EXPORT StdMidiInSetFiltered(P_BASESHELL dwpMidi, bool Value);
PREFIX void WINAPI EXPORT StdMidiInSetNoteOnZeroFilter(P_BASESHELL dwpMidi, bool Value);
/*****************************************************************************
*   FlipFlop Class access functions
*****************************************************************************/
PREFIX bool WINAPI EXPORT FlipFlopGetState(P_BASESHELL dwpFlipFlop);
PREFIX bool WINAPI EXPORT FlipFlopGetResetState(P_BASESHELL dwpFlipFlop);
PREFIX void WINAPI EXPORT FlipFlopSetState(P_BASESHELL dwpFlipFlop, bool Value);
PREFIX void WINAPI EXPORT FlipFlopSetResetState(P_BASESHELL dwpFlipFlop, bool Value);

/*****************************************************************************
*   Toggle Class access functions
*****************************************************************************/
PREFIX bool WINAPI EXPORT ToggleGetState(P_BASESHELL dwpToggle);
PREFIX bool WINAPI EXPORT ToggleGetResetState(P_BASESHELL dwpToggle);
PREFIX void WINAPI EXPORT ToggleSetState(P_BASESHELL dwpToggle, bool Value);
PREFIX void WINAPI EXPORT ToggleSetResetState(P_BASESHELL dwpToggle, bool Value);

/*****************************************************************************
*   MessageStore Class access functions
*****************************************************************************/
PREFIX void WINAPI EXPORT MessageStoreGetMessage(P_BASESHELL dwpStore, LPSTR Message);
```

```
PREFIX void WINAPI EXPORT MessageStoreGetResetMessage(P_BASESHELL dwpStore, LPSTR Message);
PREFIX void WINAPI EXPORT MessageStoreSetMessage(P_BASESHELL dwpStore, LPSTR Message);
PREFIX void WINAPI EXPORT MessageStoreSetResetMessage(P_BASESHELL dwpStore, LPSTR Message);
PREFIX void WINAPI EXPORT MessageStoreSend(P_BASESHELL dwpStore);

/*****************************************************************************
*   Display Class access functions
*****************************************************************************/
PREFIX bool WINAPI EXPORT DisplayGetMessage(P_BASESHELL dwpDisplay, LPSTR Message);
PREFIX void WINAPI EXPORT DisplaySetDisplayWindow(P_BASESHELL dwpDisplay, HWND DisplayWindow);
PREFIX void WINAPI EXPORT DisplayClearDisplayWindow(P_BASESHELL dwpDisplay, HWND DisplayWindow);
PREFIX void WINAPI EXPORT DisplayGetLastMessage(P_BASESHELL dwpDisplay, LPSTR Message);

/*****************************************************************************
*   Selector Class access functions
*****************************************************************************/
PREFIX int WINAPI EXPORT SelectorGetUpperValue(P_BASESHELL dwpSelect);
PREFIX int WINAPI EXPORT SelectorGetResetUpperValue(P_BASESHELL dwpSelect);
PREFIX int WINAPI EXPORT SelectorGetLowerValue(P_BASESHELL dwpSelect);
PREFIX int WINAPI EXPORT SelectorGetResetLowerValue(P_BASESHELL dwpSelect);

PREFIX void WINAPI EXPORT SelectorSetUpperValue(P_BASESHELL dwpSelect, int Value);
PREFIX void WINAPI EXPORT SelectorSetResetUpperValue(P_BASESHELL dwpSelect, int Value);
PREFIX void WINAPI EXPORT SelectorSetLowerValue(P_BASESHELL dwpSelect, int Value);
PREFIX void WINAPI EXPORT SelectorSetResetLowerValue(P_BASESHELL dwpSelect, int Value);
/*****************************************************************************
Table Functions
*****************************************************************************/
PREFIX unsigned WINAPI EXPORT TableGetSize(P_BASESHELL dwpTable);
PREFIX int WINAPI EXPORT TableGetValue(P_BASESHELL dwpTable, unsigned Index);
```

```
PREFIX unsigned WINAPI EXPORT TableGetNumTables();
PREFIX void WINAPI EXPORT TableGetTableName(unsigned Index, LPSTR Buf);
PREFIX void WINAPI EXPORT TableChangeTable(P_BASESHELL dwpTable, LPSTR NewName);
PREFIX void WINAPI EXPORT TableResize(P_BASESHELL dwpTable, int Value);
PREFIX void WINAPI EXPORT TableSetPoint(P_BASESHELL dwpTable, unsigned Index, int Value, bool Insert);
PREFIX void WINAPI EXPORT TableRemovePoint(P_BASESHELL dwpTable, int Value);
PREFIX void WINAPI EXPORT TableSend(P_BASESHELL dwpTable, int Index);
PREFIX void WINAPI EXPORT TableInsert(P_BASESHELL dwpTable, int Value);
PREFIX void WINAPI EXPORT TableSetMode(P_BASESHELL dwpTable, bool Value);
PREFIX P_BASESHELL WINAPI EXPORT TableLoadTable(LPSTR FileName, P_PATCH dwpParent);
PREFIX void WINAPI EXPORT TableSaveTable(P_BASESHELL dwpTable);
PREFIX bool WINAPI EXPORT TableGetModified(P_BASESHELL dwpTable);
PREFIX int WINAPI EXPORT TableGetHighestValue(P_BASESHELL dwpTable);
PREFIX int WINAPI EXPORT TableGetLowestValue(P_BASESHELL dwpTable);
/***************************************************************************
Number Store Functions
***************************************************************************/
PREFIX int WINAPI EXPORT NumberStoreGetNumber(P_BASESHELL dwpStore);
PREFIX int WINAPI EXPORT NumberStoreGetResetNumber(P_BASESHELL dwpStore);
PREFIX int WINAPI EXPORT NumberStoreGetUpperLimit(P_BASESHELL dwpStore);
PREFIX int WINAPI EXPORT NumberStoreGetLowerLimit(P_BASESHELL dwpStore);
PREFIX void WINAPI EXPORT NumberStoreSetNumber(P_BASESHELL dwpStore, int Value);
PREFIX void WINAPI EXPORT NumberStoreSetResetNumber(P_BASESHELL dwpStore, int Value);
PREFIX void WINAPI EXPORT NumberStoreSetUpperLimit(P_BASESHELL dwpStore, int Value);
PREFIX void WINAPI EXPORT NumberStoreSetLowerLimit(P_BASESHELL dwpStore, int Value);

/***************************************************************************
Trigger Functions
***************************************************************************/
```

```
PREFIX void WINAPI EXPORT TriggerSetWindow(P_BASESHELL dwpTrigger, HWND DisplayWindow);
PREFIX void WINAPI EXPORT TriggerClearWindow(P_BASESHELL dwpTrigger, HWND DisplayWindow);

/*****************************************************************************************
Random Functions
*****************************************************************************************/

PREFIX int WINAPI EXPORT RandomGenGetNumber(P_BASESHELL dwpRand);
PREFIX int WINAPI EXPORT RandomGenGetResetNumber(P_BASESHELL dwpRand);
PREFIX int WINAPI EXPORT RandomGenGetUpperLimit(P_BASESHELL dwpRand);
PREFIX int WINAPI EXPORT RandomGenGetResetUpperLimit(P_BASESHELL dwpRand);
PREFIX void WINAPI EXPORT RandomGenSetNumber(P_BASESHELL dwpRand, int Value);
PREFIX void WINAPI EXPORT RandomGenSetResetNumber(P_BASESHELL dwpRand, int Value);
PREFIX void WINAPI EXPORT RandomGenSetUpperLimit(P_BASESHELL dwpRand, int Value);
PREFIX void WINAPI EXPORT RandomGenSetResetUpperLimit(P_BASESHELL dwpRand, int Value);
PREFIX void WINAPI EXPORT RandomGenGenerate(P_BASESHELL dwpRand);

/*****************************************************************************************

Sequencer Functions
*****************************************************************************************/

PREFIX void WINAPI EXPORT SequencerGetFileName(P_BASESHELL dwpSeq, LPSTR FileName);
PREFIX void WINAPI EXPORT SequencerGetResetFileName(P_BASESHELL dwpSeq, LPSTR FileName);
PREFIX void WINAPI EXPORT SequencerGetOpenFileName(P_BASESHELL dwpSeq, LPSTR FileName);
PREFIX void WINAPI EXPORT SequencerSetFileName(P_BASESHELL dwpSeq, LPSTR FileName);
PREFIX void WINAPI EXPORT SequencerSetResetFileName(P_BASESHELL dwpSeq, LPSTR FileName);
PREFIX void WINAPI EXPORT SequencerPlay(P_BASESHELL dwpSeq);
PREFIX void WINAPI EXPORT SequencerStop(P_BASESHELL dwpSeq);
PREFIX void WINAPI EXPORT SequencerStart(P_BASESHELL dwpSeq);
```

```
PREFIX void WINAPI EXPORT SequencerEnd(P_BASESHELL dwpSeq);
PREFIX int WINAPI EXPORT SequencerNumTracks(P_BASESHELL dwpSeq);
PREFIX int WINAPI EXPORT SequencerResolution(P_BASESHELL dwpSeq);
PREFIX void WINAPI EXPORT SequencerSetTempo(P_BASESHELL dwpSeq, unsigned NewTempo);
PREFIX void WINAPI EXPORT SequencerSetResetTempo(P_BASESHELL dwpSeq, unsigned NewTempo);
PREFIX unsigned WINAPI EXPORT SequencerGetTempo(P_BASESHELL dwpSeq);
PREFIX unsigned WINAPI EXPORT SequencerGetResetTempo(P_BASESHELL dwpSeq);
PREFIX bool WINAPI EXPORT SequencerIsPlaying(P_BASESHELL dwpSeq);
PREFIX DWORD WINAPI EXPORT SequencerNumEvents(P_BASESHELL dwpSeq);
PREFIX DWORD WINAPI EXPORT SequencerCurrentPos(P_BASESHELL dwpSeq);
PREFIX void WINAPI EXPORT SequencerSetPosition(P_BASESHELL dwpSeq, DWORD NewPos);
PREFIX bool WINAPI EXPORT SequencerIsOpen(P_BASESHELL dwpSeq);
PREFIX void WINAPI EXPORT SequencerOpen(P_BASESHELL dwpSeq);
PREFIX void WINAPI EXPORT SequencerClose(P_BASESHELL dwpSeq);

#ifdef __cplusplus
}           // end C declarations for C++
#endif


#endif
```