

# From Small to Teenee

Written by Christopher Hunt of Class Action P/L, Wollongong, Australia  
email: [contact@classactionpl.com](mailto:contact@classactionpl.com)  
web: <http://www.classactionpl.com/>  
Copyright: Class Action P/L

*This article was written back in 1998 and is now out of date. However much of this document remains accurate. Please note that Teenee is now known as "ZenPlus".*

"Teenee" is a popular freeware application framework for C++ programmers who write software on PalmOS, 3Com's operating system that runs PalmPilot among others.

This article intends to explain the origins of Teenee, what it is and isn't, how it can help you as a C++ programmer and where it is going.

## Application Frameworks

For the sake of setting up terminology, here's a formal description of what a framework is: "A framework is a set of cooperating classes that make up a reusable design for a specific class of software" (Gamma et al., 1995: 26). So, Teenee is a library of classes that co-operate and form the base of a design for all types of application that can run on PalmOS. However, the previous statement does not constitute that an application framework is distinct from a "toolkit". Indeed, if the definition was left at that, Teenee would be a toolkit and not a framework. Thus, "The framework dictates the architecture of your application." (Gamma et al., 1995: 26) is quite an important statement; Teenee provides a structure to PalmOS applications based upon it.

I further regard Teenee as being a "light framework". Some frameworks tend to abstract many features in an attempt to reduce the level of understanding a programmer must have in the operating system itself; I label this type of framework a "heavy framework". Teenee is a light framework given my recognition that much of the PalmOS API is provided at a reasonable level of abstraction already. A consequence of being a light framework is that Teenee

requires prerequisite knowledge of PalmOS before it can be used effectively.

## Teenee

Teenee intends to connote "small" meaning that one of its primary goals is to be considerate of memory on PalmOS based devices; small is important given the limited amount of memory available to them.

The title of this article, "From Small to Teenee" is intended to highlight that writing applications in C for PalmOS uses *less* memory than writing in C++ and using Teenee! Why then use Teenee?

I believe that one important aspect in the approach of C++ development for embedded systems such as a PalmOS is to understand that there is a price to pay. However, I also believe that the full use of C++ and a framework such as the one that Teenee provides yields greater benefits than using C without (one of these benefits being the imposed structuring of your application as discussed with other benefits to be discussed shortly).

If you're now a little nervous of C++ and Teenee incurring an expensive memory cost, you should also understand that it consumes about 1K to 2K of memory and C++ consumes about 12K. I believe that these overheads are quite tolerable. Additionally, compiler vendors such as Metrowerks are continuing to reduce the overheads of their products. I am optimistic that the 12K overhead present in R4 and R5 of Metrowerks CodeWarrior for PalmOS will diminish to about 5K to 6K in future releases.

## Background

By now, I hope that you have a good idea of Teenee's functional scope. Just in case you haven't, minimally understand that Teenee is a C++ framework designed to help you structure your PalmOS application.

Before you think, "that's it, I've got to use Teenee in my next PalmOS based product" (which I hope you will be thinking!), I believe that it is important for you to understand how the project got started and why.

Firstly, I bought a PalmPilot and was very impressed with it. One year later, I find myself using the PalmPilot as much, if not more than when I first started using it. To me, the PalmPilot is an appliance more so that a general computing device. What I especially enjoy about PalmPilot is that it has been created from the ground up to support hand held activities; I believe that this is an important feature of the PalmPilot too often overlooked. My personal goal in computing is to provide people with solutions that they enjoy and are productive in using; PalmOS based solutions allow me to do this.

Secondly, I'm a C++ programmer predominantly. I do use other languages including Visual Basic and Java but C++ does it for me everytime! However, for PalmOS, there was a gap to be filled; that gap being the lack of any C++ productivity tools such as an application framework. Actually, I should point out that there were a few other respectable attempts at providing a C++ framework for PalmOS, but they did not appear to embody features that I consider important:

- partitioning of code into logical components;
- an attempt to adhere to Stroustrup's C++ (Stroustrup, 1997);
- use of C++ features to the full including exceptions, namespaces and templates;
- assistance in the promotion of safe

memory management;

- defensive programming by use of assertions; and
- consideration of resource, in particular, memory.

These points are not in any order of importance and build upon Stroustrup's characteristics (Stroustrup, 1997:694) of successful OO software development:

- ability to test;
- ability to port;
- ability to maintain;
- ability to extend;
- ability to reorganise; and
- ability to understand.

However, perhaps the most important point to understand is that Teenee is distributed as freeware because it personally satisfies me to do so. PalmOS is a great operating system, PalmPilot is a great device that runs it and I may be able to contribute toward the success of both by distributing Teenee.

## Nuts and Bolts

The simplest useful example that I can give is the old favourite "Hello World!" one. I state "useful" given that I could provide an example without having any user interface functionality. However, in this instance you would not be able to get any visual confirmation of it working. The HelloWorld example can be found in the distribution of Teenee so don't bother typing it in again!

Before we start with code, the following UML (Unified Modelling Language) class specification diagram depicts the classes that we will be using of the Teenee namespace:

AbstractApplication provides essential behaviour for PalmOS to be able to launch and manage an application process. GeneralForm provides essential form management for a

PalmOS application.

AbstractApplication and GeneralForm both have private event handlers named mEventHandler. An event handler provides a mechanism for processing PalmOS events. Each event handler class inherits from EventHandler. The class diagram illustrates a common strategy within Teenee to use composition as distinct from inheritance to adorn behaviour to a class. Composition promotes higher reuse and more manageable components among other benefits (Gamma et al., 1995: 18). All too often, we have seen class libraries that use inheritance to add behaviour thus confusing a programmer into understanding a given abstraction is many things i.e. it has more than one "type" (Gamma et al., 1995: 13). The following UML class specification diagram illustrates the relationships between the Teenee namespace and HelloWorld:

Application inherits from AbstractApplication and depends on an instance of GeneralForm for the rendering and handling of a form. ApplicationRunner is a template instantiation of Application which manages PalmOS entry points.

Now lets look at a code sample to add further comprehension to the diagrams. The HelloWorld example consists of a header file and an implementation file. Here are the contents of the header file:

```
#include <AbstractApplication.h>
#include <GeneralForm.h>

class Application :
public Teenee::AbstractApplication
{
public:
Application()
    Throw_(std::bad_alloc);

virtual void
    SysAppLaunchCmdNormalLaunch(
        const Ptr inCmdPBP,
```

```
Word inLaunchFlags
);
```

```
private:
Teenee::GeneralForm mMainForm;
};
```

Simply put, our header file content declares the outline of an application with one form and an entry point for PalmOS with SysAppLaunchCmdNormalLaunch. The core of Teenee functionality is provided with AbstractApplication and GeneralForm. "Abstract" infers that the class is not intended for direct instantiation while "General" means that the class can be used directly though is often subclassed to perform anything really useful. Teenee headers are conventionally included using the system form i.e. between < and > characters. Another convention is that using declarations never appear in header files though often in implementation files. This is because the inclusion of using in a header assumes knowledge of how the implementation code uses namespaces; thus, we do not do it.

Our implementation code looks like this:

```
#include "Application.h"
#include "Application.rsrc.h"
#include "ApplicationRunner.cpp"

using namespace std;
using namespace Teenee;

const ULong
AbstractApplication::kAppFileCreator
    = 'Tiny';
const Word
AbstractApplication::kAppVersionNum
    = 0x01;
const Word
AbstractApplication::kAppPrefID
    = 0x00;
const Word
AbstractApplication::kAppPrefVersion
Num
    = 0x01;
```

```

Application::Application()
Throw_(std::bad_alloc) :
    MainForm(
        GetEventHandler(),
        HelloMainForm
    )
{}

void
Application::SysAppLaunchCmdNormalLa
unch(
        Ptr inCmdPBP,
        Word inLaunchFlags
    )
{
#pragma unused(inCmdPBP)
#pragma unused(inLaunchFlags)

::FrmGotoForm(HelloMainForm);

EventLoop();
}

DWord PilotMain(
    Word inCmd,
    Ptr inCmdPBP,
    Word inLaunchFlags
)
{
Teenee::ApplicationRunner<Applicatio
n>
    theAppRunner;

    return theAppRunner.Run(
        inCmd,
        inCmdPBP,
        inLaunchFlags
    );
}

```

Application.h is our header and Application.rsrc.h contains symbol definitions for our PalmOS resources mainly used for describing user interface elements (forms, menus, strings etc.). ApplicationRunner.cpp contains template code to handle the instantiation of our application object and the various entry point requirements of PalmOS e.g. application launch and a find request.

ApplicationRunner.cpp is typically copied from the Teenee folder into your project folder and then modified in accordance with how you want your application to respond to PalmOS entry points.

AbstractApplication requires that you provide important information such as the file creator and application's version so that the application can be properly registered with PalmOS.

The constructor of our Application class provides an event handler to the GeneralForm member, MainForm and the symbol value of the HelloMainForm resource. Event handlers are instances of EventHandler and implement the "Chain of Responsibility" design pattern. To quote from Design Patterns, chain of responsibility avoids, "...coupling the sender of a request [PalmOS] to its receiver [Teenee objects] by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it." (Gamma et al., 1995: 223). Teenee provides AppEvtHandler and FormEvtHandler to handle application and form events respectively. Application events concern themselves with loading forms and setting a chain of responsibility to the chain that corresponds to an active form (there can be multiple chains given multiple form hierarchies). Form event handling complements AppEvtHandler chain of responsibility handling and form opening and closing. Thus, Teenee provides you with an application that can correctly load forms and display them without you having to do anything.

SysAppLaunchCmdNormalLaunch is called when PilotMain receives the SysAppLaunchCmdNormalLaunch command as the inCmd parameter. Dispatch of this application's method is provided by ApplicationRunner. I won't include the code for ApplicationRunner as it is very similar across all projects and you can easily see the code for

yourself; suffice to state that ApplicationRunner is responsible for dispatching based on the value of inCmd. Our implementation of application launch causes the focus of the main form and starts event loop processing by calling AbstractApplication's EventLoop method. EventLoop handles all event dispatch requirements as a well behaved PalmOS citizen should.

It may seem like an awful lot of code to display "HelloWorld", but most solutions will require more than just displaying some text. The point here is that what you have seen so far forms the basis of even the most complex Teenee based application. You can write the HelloWorld sample without using C++ or Teenee much more simply using C. However, if you use Teenee for real world solutions, I hope that you'll appreciate the forced partitioning of code into more manageable components, and the quality of result given that you do not have to recode event handling and PalmOS entry points. Indeed, Teenee should allow you to focus on the problem in hand and not mundane PalmOS requirements.

### More Code

A MemoPad example which is part of the Teenee distribution provides a stronger starting point for real PalmOS applications. The MemoPad example is largely a rewrite of the standard sample code which is part of the PalmOS SDK from 3Com. Teenee MemoPad illustrates database IO, safe memory management and the partitioning of code.

The promotion of safe memory management is an important feature of Teenee. Sentries (Stroustrup, 1997:624) provide the guaranteed release of important memory constructs. Here's a little code illustrating the use of a sentry in the course of writing memory to a database:

```
Ptr theRecP = static_cast<Ptr>(
```

```
        ::MemHandleLock(theRecH)
        );
{
MemHandleUnlockSentry
    theRecHUnlocker(theRecH);

Char theZeroChar = 0;
ThrowIfErr_ (::DmWrite(
                theRecP,
                0,
                &theZeroChar,
                sizeof(theZeroChar)
            ));
}
```

The preceding code locks a handle in memory using ::MemHandleLock and then declares a new scope between the { and } characters. A MemHandleUnlockSentry is instantiated on the stack so that when it goes out of scope (after the ::DmWrite), the handle is automatically unlocked. The advantage of using a sentry in our example is the guaranteed unlocking of the handle no matter whether ::DmWrite succeeds or throws an exception.

A common question that I've been asked by Teenee programmers is along the lines of why doesn't the constructor of a sentry perform the ::MemHandleLock (or whatever is appropriate for that type of sentry)? The simple answer is that there can be multiple ways in which a resource can be locked or allocated but generally only one way for it to be unlocked or deallocated respectively. For example, ::DmQueryNextInCategory can also lock a handle as well as ::MemHandleLock. Future releases of Teenee might reintroduce a Guardian abstraction which I now regard as a stronger, more possessive type of sentry. The name, "Guardian" was invented by myself (as far as I know!) for generally describing what Stroustrup refers to as a Sentry; so I renamed my guardians to sentries for the sake of consistency in terminology.

I recommend a practice of whenever you

lock, allocate or do something for which there is a complementary function, write the call for the complementary function immediately so that you don't forget! Sentries in Teenee provide most of the complementary functions you should need for PalmOS programming. Teenee's distribution also provides the `std::auto_ptr` class (Stroustrup, 1997:367) which is a very commonly used sentry in ANSI C++ programming.

### Getting Started

Firstly, you will need to download Teenee from our website:

<http://www.classactionpl.com/Teenee>

If you have the Metrowerks CodeWarrior for PalmOS product, you will find it easiest to download the self extracting archive (.sea). This archive includes CodeWarrior projects that clearly describe the build requirements of Teenee.

If you do not have Metrowerks CodeWarrior (and this appears to be the majority of the PalmOS development community), then download the zip file instead. Figuring out the dependency hierarchy of Teenee shouldn't be difficult given that there are relatively few components. Essentially, there is one file per class (although strongly related classes may all appear in the one file), and there is a header file to each corresponding implementation file (where appropriate). The filenames of Teenee components generally correlate to their defining class names.

Because I've attempted to adhere to standard C++ coding practices and resources, you should have little problem in using Teenee with GNU or other PalmOS C++ compilers. Both archives contain complete source code for Teenee, a sample derived from 3Com's MemoPad application and our HelloWorld

sample. The MemoPad program has almost been completely rewritten from 3Com's original and is totally supportive of Teenee. As a courtesy, please understand that 3Com Inc. own the copyright of the MemoPad program, Class Action owns the copyright of Teenee and Modena Software Inc. own the copyright of the contents of "Teenee Std" (fill in code for the ANSI C++ library in lieu of one becoming available for PalmOS). Please respect the copyright notices enclosed in the files which includes not changing them.

Important disclaimer: Teenee is provided as is. Class Action Pty. Ltd. absolutely accepts NO responsibility for ANY consequences of using the code.

At the time of writing this article, Teenee is in a "beta" state. This means that Class Action does not recommend you using it for production code today. However, the code is freeware and there a few restrictions on its use. You have the source, so you have the resources you require to fix problems should they occur. It may encourage you to know that very few problems have been reported since the alpha release in June, 1998.

### The Future

Upon receiving Metrowerks CodeWarrior for PalmOS R5, I shall work upon a production worthy release of Teenee V1.0.0 which will be the present Beta version available from my website. The Beta version incorporated the following changes that are reflected by some parts of this article:

- small bug fixes;
- constructors that throw exceptions guarantee that their corresponding object is destructed;
- Guardians have been renamed as "Sentries" in line with Stroustrup's terminology. Guardians may appear again in the future as a stronger type of sentry. For example, a guardian may include, as part of its

construction, the opening of a database while its destructor will close the database. A sentry is simply be responsible for closing a database on the presumption that it has already been opened; and

- given that namespaces are being used, there is no need for class names to receive the “L”, “C” and “U” prefixes as they used to. Consequently, the prefixes have been removed thus promoting clearer naming.

In the course of performing work outside of Teenee, I have created a C++ toolkit (as distinct from a framework) primarily aimed at portable Win32 and MacOS development. I may complement Teenee with some of Classee's components. For example, the “Observer” design pattern (Gamma et al., 1997:293) and reference counted object classes among a few other utilities. Other than this thought, I'm open to suggestions!

### **How Can You Help?**

You can help in the development of Teenee by providing feedback. I have received very positive feedback from users of Teenee, but not enough!

I've found that I get feedback enough when there is something to complain about, so I can only assume that Teenee is perfect for everybody given the little feedback there has been!

Feedback can include:

- what application(s) you are using Teenee for;
- what features do you think are missing; and
- how Teenee can be improved.

I'm especially interested in your efforts to use Teenee under compilers other than Metrowerks CodeWarrior.

I endeavour to answer e-mails promptly, so roll them in! Additionally, by contacting me, I shall do my best to inform you of Teenee's

new releases.

If you write application based on Teenee, I would really appreciate a mention of it in your application's “Info” dialog. Having the ability to cite uses of our code helps us in our paid work!

### **About the Author**

Christopher Hunt is the Director of Class Action Pty. Ltd which is based in Wollongong, NSW, Australia. Christopher lives with his partner, Tina. They both live by the beach and countryside and spend a fair bit of time enjoying them when not working. When they don't do either of these things, they can mostly be found falling out of aircraft!

Christopher takes a special computing interest in the development of real time and fault tolerant applications.

Class Action is presently developing product mainly in the domains of Computer Telephony Integration and Telecommunications. However, Class Action's main goal is to provide solutions that people enjoy and are productive in using. Class Action intends to publish its class libraries used in the development of application as software for all to enjoy. Class Action's policy is based on the recognition that to take from the industry, they should contribute something also.

### **Bibliography**

- Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) *Design Patterns - Elements of Reusable Object-Oriented Software*. Reading, Massachusetts, Addison-Wesley.
- Stroustrup, B. (1997) *C++ Programming Language - Third Edition*. Reading, Massachusetts, Addison-Wesley.