Proposed XDK Enhancements

For FreeHand 10

Introduction

This document contains three separate proposals for XDK Enhancements in the next version of FreeHand. The three proposals represent differing levels of effort; the first proposal would be relatively quick and painless, the second proposal would require more effort, and the third proposal might require multiple engineers. Each proposal is a simple overview that includes a summary, a list of tasks, pros and cons, and, if needed, examples that further illustrate the intent of the proposal.

Background

MOA, the FreeHand XDK and the FreeHand Xtras API were originally designed to allow third-party developers to extend the capabilities of FreeHand, thus adding value to the product and allowing others to profit on FreeHand's success. Some of FreeHand's largest customers, such as Knight Ridder and American Greetings, are marginally successful in using the XDK to make it easier for their artists to work more efficiently. However, unlike identical efforts by Quark (Xpress) and Adobe (Illustrator, Photoshop), the FreeHand XDK has failed to produce a significant number of third-party companies and products.

The FreeHand XDK has also caused internal problems. Existing Xtras are complex, poorly written, and suffer from frequent problems in the API. And although Xtras would ideally be a great starting point for new engineers, the complexity and poor design of the API is not conducive to such usage.

In short, the existing API suffers from a lack of direction and a level of priority that ranks well below many typical cycle-to-cycle engineering tasks. In order to make real improvements to the API, we need to answer a few questions first. What is the purpose of the API and the XDK? Who are our target developers? How much time can we devote to improvements and maintenance in the future? What is the balance of external vs. internal usage of the API? Regardless of the answers to these questions, we can use them to build a more practical and useful XDK.

Proposal I: Project Maintenance

Summary

The intent of Proposal I is to integrate all FreeHand Xtras, including those dating back to FreeHand 5, into the build and source control environments. Adding TRACE statements to all API implementations and preparing to Carbonize Xtras are also included. These tasks should be treated as requirements before commencing the next development cycle.

Tasks

- Find source code and resources for all shipping FreeHand Xtras.
- Update each project to match current build policies and make sure that each project successfully compiles, links, and runs.
- Place all Xtra assets into source control.
- Add TRACE statements to all API implementations in FreeHand to aid debugging.
- Prepare Xtras for Carbonization: assess the implications to engineering and QA.

Pros and Cons

The tasks in this proposal should not require significant amounts of time and resources, although they are somewhat tedious. This proposal does not address any of the problems in the API and XDK.

Rough time estimate: 2-3 weeks.

Proposal II: Minor Cleanup

Summary

Proposal II outlines a sweep through the API that is roughly equivalent to a “physical”. Many of the APIs 1) do not perform basic error checking, 2) do not correctly handle reference counting, and 3) contain outdated and improperly functioning code that has not been tested. These are the types of problems that should be sought out and fixed. Other problems covered here included missing functionality (such as the ability to remove a layer) and the API’s inability to allow Xtras to perform automated testing (lack of support for idle processing and timers).

The extent of the tasks performed for this proposal will likely depend solely on the availability of time and resources.

Tasks

- Perform the tasks outlined in Proposal I.
- Examine each API implementation for proper error and exception handling, reference counting, and usage of FreeHand code. Other problems should be sought out and fixed as needed.
- Fill key functionality holes, such as deleting layers.
- Add support for automated testing. Xtras hog CPU time and system resources while running, preventing them from performing actions for extended periods of time.
- Begin developing Xtras designed to test both FreeHand and the stability of the API.

Pros and Cons

These tasks will require more time and resources than those in Proposal I. If executed properly and with care, the clean sweep should improve the stability and usefulness of the API and raise the confidence level of the engineers using it. However, this proposal still does not address the primary underlying problems in the API and XDK.

Rough time estimate: 2 months.

Proposal III: A New XDK

Summary

Proposal III outlines the creation of a new API and associated XDK. The goal is to create an XDK that has the characteristics listed below.

Minimal But Complete - The API should allow developers to successfully meet their requirements, yet it should not be cluttered with useless or "hacked" APIs. The key to this characteristic is understanding the developer's requirements.

Stable - There should be a secure boundary between FreeHand and Xtras, and the implementation of the API should allow for unanticipated usage by third parties.

Maintainable - The internals of the API and its usage by Xtras should be straightforward and well documented. Modifications to the API or to Xtras should be easy and consistent in all future FreeHand releases. Proprietary technology should be used with care.

Flexible - Inevitably, APIs and Xtras change from release to release. The API should allow for such transitions to occur gracefully and within the bounds of the original API/XDK design.

The first and most important step in performing such a task is to determine the direction and purpose of the XDK. The layout and capabilities of the API should reflect the needs of the target audience, which may or may not include FreeHand engineers, commercial Xtra developers, or corporate "in-house" developers. The remaining steps are quite straightforward. The implementation of an external API is relatively trivial and non-intrusive to the application code.

Tasks

- Determine the target developer audience.
- Design an API and underlying architecture that reflects the needs of that audience.
- Implement the API.
- Provide facilities for testing the API.
- Create an associated XDK, including documentation and source code

Pros and Cons

This proposal represents the best total solution for the existing problems in the API and XDK. It allows us to replace a troublesome, difficult to maintain API with a more modern, practical API. However, it will require significant amounts of time and resources. Further, we will need to address the current and future state of all existing Xtras.

Rough time estimate: 6 months.

Examples

The examples below help illustrate the intent of this proposal.

Improved Usage

Consider the following code snippets. The first snippet uses conventions from the existing XDK. The second snippet uses conventions from a hypothetical new XDK.

```
////////////////////////////////////////////////////////////////////////
// Snippet 1

STDMETHODIMP_(MoaError) CMyTool_IMoaFhToolItem::GetToolIcon(
      CPIconHan FAR * pIcon
)
{
moa_try

      // variable declarations
      PIMoaCallback           pCallback;
      PIMoaFhCallback         pFhCbk;
      PIMoaCPGraf             pCPGraf;
      PIMoaCPRes              pCPRes;
      XtraResourceCookie      myCookie, saveCookie;

      // variable initialization
      pCallback = pObj->pCallback;
      pFhCbk = pObj->pFhCbk;
      pCPGraf = pFhCbk->GetCPGraf();

      ThrowErr( pCPGraf->QueryInterface(&IID_IMoaCPRes, (PPMoaVoid) &pCPRes) );

      // access resources
      ThrowNull( (myCookie =
                  pCallback->MoaBeginUsingResources(gXtraFileRef, &saveCookie)) );

      // load the icon
      ThrowNull( *pIcon = pCPRes->CPResLoadIcon(myCookie, CKochTool_TOOL_ICON) );

moa_catch

      MOA_DEBUGSTR( ("Exception thrown at %s.", moa_get_exception_location) );

moa_catch_end

      // release
      if (myCookie)
            pCallback->MoaEndUsingResources(gXtraFileRef, saveCookie);
      if (pCPRes)
            pCPRes->Release();

moa_try_end
}

////////////////////////////////////////////////////////////////////////
// Snippet 2

long XTRA_CALLBACK CMyTool::GetToolIcon(void ** ppIcon)
{
      HICON hIcon = NULL;
      long  lErr = 0;

      try {

            // load the icon
            hIcon = ::LoadIcon(m_hInstance, “MyToolIcon”);
            if (hIcon) {
                  *ppIcon = (void *) hIcon;
```

```
                    lErr = 1;
              }

        } catch (…) {

              *ppIcon = NULL;
              lErr = -1;
        }

        return(lErr);
  }
```

Note the use of standard C++ construucts, the lack of macros, and the use of standard OS functions. As you can see, there is plenty of room for improving the developer's experience with the usage of the API.

Better API Layout

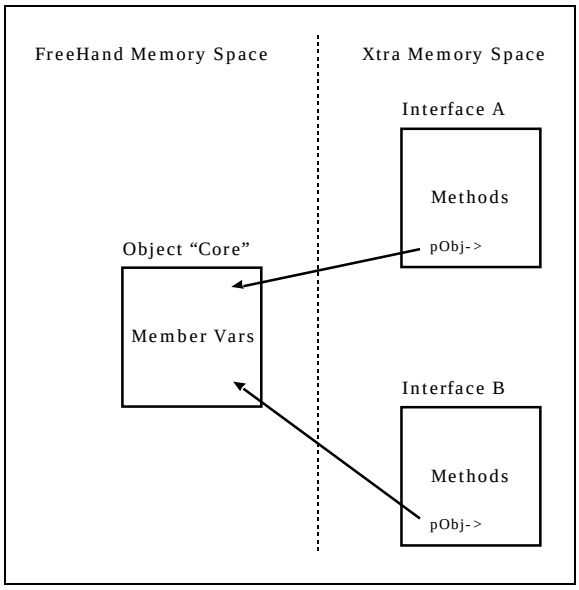
Consider the following API layouts for layer functionality. The first layout is from the existing API. The second layout is from a hypothetical new API.

```
////////////////////////////////////////////////////////////////////////
// Layout 1

IMoaFhCallback (includes over 300 other methods in the same interface)
      …
      FHINewLayer()
      FHIGetLayerName()
      FHIGetNumLayers()
      FHISetCurrentLayer()
      FHIGetCurrentLayer()
      FHISetLayerFlags()
      FHIGetLayerFlags()
      …

////////////////////////////////////////////////////////////////////////
// Layout 2

LayerManager
      Create()
      Reorder() *
      Delete() *
      GetNum()
      GetNth()
      SetCurrent()
      GetCurrent()
      SetName() *
      GetName()
      SetFlags()
      GetFlags()
```

Items marked with a (*) indicate functionality that is not currectly available in the API. The second layout reflects a much more intuitive object model with easy to use and remember method names. With a more intelligent API layout, we can provide a more practical and powerful XDK.

Simple Architecture

Consider the following architecture diagrams. The first diagram illustrates the structure of a MOA object from the existing XDK. The second diagram illustrates the structure of a C++ object from a hypothetical new XDK.



Layout of a MOA Object

Layout of a C++ Object

The second diagram represents an architecture that is far more straightforward yet equally functional.