An introduction to using the PageMaker API.

The PageMaker Application Programming Interface (API) is made up of interface classes. Each interface is a group of functions that access a common feature. In this introduction to the API we will use a basic interface to demonstrate how you will use the component interfaces in the Plug-ins that you create.

The Interface Manager and the Component Interfaces are C++ classes. To the programmer who is not versed in C++, these can be approached as a 'group of functions'. The 'group of functions' approach to C++ classes is an oversimplification of C++ classes, but for the purposes of using the PageMaker API it is a usefull description.

Both in using the Interface Manager and in using the individual Component Interfaces, you will use the format `classPointer->memberFunction()`, where classPointer is the pointer to the class you are using and memberFunction is the name of the function within that class.

The interface that we will use is the CIBasic interface. This interface includes many of the functions that you will use in all of your Plug-ins and are not tied to a specific feature in the PageMaker Application. These functions will let you allocate and free memory, convert between screen and world coordinate systems, and alter your Plug-in's properties while it is running.

The CIBasic interface is defined in the file CIBasic.h, which you will find in the "Includes" folder in this SDK.

```
class CIBasic : public CIInterface
{
public:
      //    Memory functions. It is recommended that you use
      //    these function to manage memory.
      virtual void *PMMemAlloc(unsigned long) = 0;
      virtual void PMMemFree(void *) = 0;
      virtual void *PMMemRealloc(void *, unsigned long) = 0;

      // More functions are included in the CIBasic.h file
      //    than are shown here in this example.
...
};
```

The CIBasic class is accessable through the interface manager, covered in the CIInterfaceManager documentation, so before you use the CIBasic functions you must 'acquire' the CIBasic interface.

```
result = pIntfMgr->AcquirePMInterface(PMIID_BASIC, (void *)&pBasic);
```

The AcquirePMInterface function return is a PMErr type. If the interface has been aquired the return is CQ_SUCCESS. If the interface was not acquired, the return value is an error code. The AcquirePMInterface function receives two values, the ID (or name) of the interface, and a pointer that you will use to access the interface. The pointer (pBasic in the example above) is how we get to the member functions within the class.

```
pMyBlock = pBasic->PMMemAlloc(myBlockSize);
// the pMyBlock pointer now points to a block of memory allocated by PageMaker.
pBasic->PMMemFree(pMyBlock);
// the pMyBlock pointer has been freed
```

Normally you wouldn't allocate memory, just to free it in the very next statement, but the example above does show how to use the class pointer to call it's functions.

Releasing an interface allows the resources that it uses to be freed. You will use the Interface Manager to release the interface:

```
pIntfMgr->ReleasePMInterface(pBasic);
```

This method of acquiring an interface, using it, and releasing it is how Plug-ins can get to all of the interfaces available in the PageMaker application, and custom interfaces created by other Plug-ins (although the form for custom interfaces is a little different, see below.) In most cases, acquiring an interface just before it is used and then releasing it immediately will allow you to keep the scope of the interface to a minimum. This will make keeping track of the interfaces simpler. In the following example, we want to allocate memory, so we acquire the CIBasic interface, and then allocate the memory. Releasing the interface immediately frees the pBasic pointer, but the memory we allocated with the interface remains available.

```
result =  pIntfMgr->AcquirePMInterface(PMIID_BASIC, (void *)&pBasic);
// Error checking code skipped in this example to make the example as simple as
possible.
pMyBlock = pBasic->PMMemAlloc(myBlockSize);
pIntfMgr->ReleasePMInterface(pBasic);
```

When we want to free the memory, we need to acquire the interface again.

```
result =  pIntfMgr->AcquirePMInterface(PMIID_BASIC, (void *)&pBasic);
// Error checking code skipped in this example to make the example as simple as
possible.
pBasic->PMMemFree(pMyBlock);
pIntfMgr->ReleasePMInterface(pBasic);
```

Many of the interfaces include methods that are setup or cleanup methods and must be called before or after other methods. These methods are noted in the reference documents of those interfaces. An example of an interface with both a setup and finishing method is the CISaveImage interface. This interface allows your Plug-in to specify a graphic image in the current publication, and export it to a different image type. Using this interface would look something like this;

```
PMXErr      result;
CISaveImage *pImSave;
PMOBJ_REC   myObj;

//... code to get the object record for the graphic skipped...
//See CIObjectAccess documentation

result = pIntfMgr->AcquirePMInterface(PMIID_IMSAVE, (void *)&pImSave);
result = pImSave->Setup(myObj);

// image attributes would then be set, & the image exported.
result = pImSave->SaveIt();
result = pImSave->Finished();
pIntfMgr->ReleasePMInterface(pImSave);
```

The code sample above would not handle error conditions, since each value stored in result is being thrown away rather than checked, but the code sample does show the order of events when using an interface that has a Setup() and a Finished() method.

Custom Interfaces

A Plug-in has the option of acquiring interfaces from other Plug-ins. To acquire an interface from another Plug-in the Plug-in that is supplying the interface must be installed and must, at the time the interface is requested, have the interface registered with PageMaker. The Plug-in that is acquiring the custom interface must know the exact name of the interface and its signature to use it.

Acquiring a custom interface is accomplished with the same method that any other interface is acquired with, but instead of an ID the Plug-in must supply the interface's name:

```
void  *theInterface;

pIntfMgr->AcquirePMInterface("TheCustomInterface", &theInterface);
```

Once the interface has been acquired it is treated as a pointer to a function.

```
infoBack = (theInterface)(firstArg, secondArg);
```

For information on creating a custom interface see the Custom Interface information in the Topics section.