Working with P/XI Stacks

When working inside the P/XI UI within the context of working with stacks, there are four kinds of objects you will be working with:

RADynFileFmtIO works as a handle to a project file. It does everything related to accessing and editing the DFF items of a DFF file. When working with a stack, this class is rarely used except for giving a method access to an open project file, or retrieving something referred to by a property of an object.

StackWindow is the window that displays a stack. It must be accessed whenever we want to switch to a different card or manage its cache.

PXIObject is the most commonly used kind of object. It can hold a card, background, stack, part, report template, or report item. It contains many of the methods we are going to use, and it's faster than using a dictionary for accessing individual properties of an object.

Dictionaries can be generated from PXIObjects. You will want to use them if you're accessing more than a couple properties at once, since it's faster to translate an object's internal data into a dictionary once than to scan it multiple times to read multiple properties.

Grabbing Ahold of the File Itself

To start with, you need to get access to an RADynFileFmtIO or a StackWindow, or be given one. You can use the standard RealBasic Window function to scan through windows, looking for a StackWindow. Or, you can use the GetStackWindow function:

```
function GetStackWindow(projectFile as folderItem, stackID as
   integer, openIfNot as boolean) as StackWindow
```

This will get the StackWindow of any stack in any open project file. If `true` is passed in the `openIfNot` parameter, the StackWindow will be opened if it isn't already; otherwise, `nil` will be returned for any stack that isn't open.

You can also grab ahold of a ProjectWindow by looking through all the open windows for one. Both StackWindows and ProjectWindows have a property called `MyFile`, which is the RADynFileFmtIO used to read or write to the file. StackWindows have a property called `MyStackID`, which is the ID of the stack the window is displaying.

Here are the most useful methods in the StackWindow class:

```
sub ChangeMyBkgndObject(bkgndId as integer, theObject as
   PXIObject)
sub ChangeMyCardObject(cardId as integer, theObject as PXIObject)
sub ChangeMyStackObject(theObject as PXIObject)
sub DoPageSetup()
sub DumpCaches()
function IDofBkgndNumber(bkgndNumber as integer) as integer
function IDofCardNumber(cardNumber as integer) as integer
function MyBkgndObject(bkgndId as integer) as PXIObject
function MyCardObject(cardId as integer) as PXIObject
function MyStackObject() as PXIObject
function NumberOfBkgndId(bkgndId as integer) as integer
function NumberOfCardId(cardId as integer) as integer
sub PaintCard(g as graphics, cardId as integer, baseX as integer,
   baseY as integer, colorMode as integer, excludeStack as
   boolean, excludeBackground as boolean, excludeCard as boolean)
sub SwitchToCard(cardNumber as integer)
```

The StackWindow itself won't be used to access the stack itself, however, unless you want to use one of its methods. What we mainly want out of a StackWindow is the RADynFileFmtIO it uses to access the project file, and the ID number of the stack to work on. We get these from the `MyFile` and `MyStackID` properties of the StackWindow.

Once we have the RADynFileFmtIO and the stack ID, we have unlimited access to the objects in the project file.

Working with Stacks, Cards, Parts, and So On

One of the main things we'll be working with are PXIObjects. We use PXIObjects to read and write all the stacks, backgrounds, cards, parts, report templates, and report items in a project file. There are three methods we use for manipulating objects in a project file:

```
function GetObject(projectFile as RADynFileFmtIO, objectType as
   string, objectID as integer) as PXIObject
sub SetObject(projectFile as RADynFileFmtIO, objectType as string,
   objectID as integer, object as PXIObject)
sub DeleteObject(projectFile as RADynFileFmtIO, objectType as
   string, objectID as integer)
```

`ObjectType` is either `"card"`, `"background"`, `"stack"`, or `"template"`.

A blank PXIObject, with no properties and no parts, can also be created with the `new` keyword.

Once we have a PXIObject, we can change its properties or parts.

The PXIObject class acts kind of like a dictionary. Using a PXIObject is much faster than using a dictionary, because you don't have to convert objects to dictionaries and back again every time you do something. In addition to having dictionary-like methods, this class has some P/XI-specific methods.

The following methods work with properties of the card, background, stack, or template:

```
function Count() as integer
function GetDictionaryValue(key as string) as dictionary
function GetPict(fileHandle as RADynFileFmtIO) as picture
function GetValue(key as string) as variant
function HasKey(key as string) as boolean
function Key(i as integer) as string
sub Remove(key as string)
sub SetValue(key as string, value as variant)
```

These all work similarly to a dictionary's properties and methods. Keys are always strings. A couple of these methods deserve special mention. The GetDictionaryValue method returns a property as a dictionary, if it is a dictionary; this is safer than assuming it's a dictionary, and less tedious than going through the motions of making sure it's a dictionary. The GetPict method returns the picture property of the object as a picture.

The following methods work with parts on a card, background, or stack, or items on a report template:

```
function CountParts() as integer
function GetPart(partNumber as integer) as PXIObject
function HasPart(partNumber as integer) as boolean
```

```
sub RemovePart(partNumber as integer)
sub SetPart(partNumber as integer, value as PXIObject)
```

These work similarly to the methods for working with properties. The PXIObject class also has methods for translating one type of part identity to another, in case you need to work with parts by ID or name rather than number:

```
PartIDtoName(partID as integer) as string
PartIDtoNumber(partID as integer) as integer
PartNameToID(partName as string) as integer
PartNameToNumber(partName as string) as integer
PartNumberToID(partNumber as integer) as integer
PartNumberToName(partNumber as integer) as string

PartIDofTypeToPartNumberOfType(partType as integer, partID as
   integer) as integer
PartNameOfTypeToPartID(partType as integer, partName as string) as
   integer
PartNameOfTypeToPartNumber(partType as integer, partName as
   string) as integer
PartNameOfTypeToPartNumberOfType(partType as integer, partName as
   string) as integer
PartNumberOfTypeToPartID(partType as integer, partNumber as
   integer) as integer
PartNumberOfTypeToPartName(partType as integer, partNumber as
   integer) as string
PartNumberOfTypeToPartNumber(partType as integer, partNumber as
   integer) as integer
```

The Part*OfType methods are used whenever you're dealing with specifically parts of a certain type, rather than parts in general. To work with buttons, pass 1 for partType. To work with fields, pass 2 for partType. For example, if you want to find the part ID of field number 8, use this:

```
id=PartNumberOfTypeToPartID(2,8)
```

The following will return the button number of a button named "Rebecca":

```
fieldNum=PartNameOfTypeToPartNumberOfType(1,"Rebecca")
```

You may be surprised to hear that the methods for getting or setting parts will not be used often. The reason for this is because it is faster to read or write the properties of a part from its parent object than to separate the part from its parent object and manipulate the properties there. Here are the methods for working with properties of parts:

```
function PartCount(partNumber as integer) as integer
function PartGetDictionaryValue(partNumber as integer, key as
   string) as dictionary
function PartGetPict(partNumber as integer, fileHandle as
   RADynFileFmtIO) as picture
```

```
function PartGetValue(partNumber as integer, key as string) as
   variant
function PartHasKey(partNumber as integer, key as string) as
   boolean
function PartKey(partNumber as integer, i as integer) as string
sub PartRemove(partNumber as integer, key as string)
sub PartSetValue(partNumber as integer, key as string, value as
   variant)
```

These work exactly like the methods for accessing properties of the object itself.

Properties that cannot be represented as a string or number are returned as dictionaries. This is what GetDictionaryValue is used for. If the property is a rectangle, the dictionary will have `left`, `top`, `right`, and `bottom` values; if the property is a point, the dictionary will have `x` and `y` values; if the property is a reference to something else in the project file, the dictionary will have `symbol` and `id` values (which are passed on to the RADynFileFmtIO).

If we're working with many properties at once, we'll want to convert a PXIObject into a dictionary and back (if they've been changed). For working with more than two or three properties, a dictionary is faster since the data is scanned only once. Here are the methods for converting a PXIObject or one of its parts to a dictionary and back:

```
function GetAsDictionary() as dictionary
function GetPartAsDictionary(partNumber as integer) as dictionary
sub SetToDictionary(dict as dictionary)
sub SetPartToDictionary(partNumber as integer, dict as dictionary)
```

Once turned into a dictionary, properties appear under keys that are strings, and parts appear as dictionaries under keys that are integers. (This is the reason why keys of PXIObjects are always strings, by the way.) A part may be retrieved quite easily using the ExtractDictFromDict method:

```
function ExtractDictFromDict(parentDictionary as dictionary, key
   as variant) as dictionary
```

ExtractDictFromDict can also be used for accessing rectangles, points, and DFF references in PXIObjects converted to dictionaries.

Identity Crisis

PXIObjects can only be read or written by ID number, and parts can only be accessed by index number. In order to access something by some other form of identification, such as name, you must convert it to an ID number (for an object) or index number (for a part). The PXIObjectIdentityLib module has tons of methods for converting from just about any form of identification to any other. Some of these methods require stack objects or RADynFileFmtIO's to work.

```
function pxi_BkgndIDtoName(theFile as RADynFileFmtIO, bkgndId as
   integer) as string
function pxi_BkgndIDtoNumber(StackObject as PXIObject, bkgndId as
   integer) as integer
function pxi_BkgndNameToID(theFile as RADynFileFmtIO, StackObject
   as PXIObject, bkgndName as string) as integer
function pxi_BkgndNameToNumber(theFile as RADynFileFmtIO,
   StackObject as PXIObject, bkgndName as string) as integer
function pxi_BkgndNumberToID(StackObject as PXIObject, bkgndNumber
   as integer) as integer
function pxi_BkgndNumberToName(theFile as RADynFileFmtIO,
   StackObject as PXIObject, bkgndNumber as integer) as string
function pxi_ButtonIDtoNumber(theObject as PXIObject, buttonId as
   integer) as integer
function pxi_ButtonNameToNumber(theObject as PXIObject, buttonName
   as string) as integer
function pxi_ButtonNameToPartID(theObject as PXIObject, buttonName
   as string) as integer
function pxi_ButtonNameToPartNumber(theObject as PXIObject,
   buttonName as string) as integer
function pxi_ButtonNumberToPartID(theObject as PXIObject,
   buttonNumber as integer) as integer
function pxi_ButtonNumberToPartName(theObject as PXIObject,
   buttonNumber as integer) as string
function pxi_ButtonNumberToPartNumber(theObject as PXIObject,
   buttonNumber as integer) as integer
function pxi_CardIDtoName(theFile as RADynFileFmtIO, cardId as
   integer) as string
function pxi_CardIDtoNumber(StackObject as PXIObject, cardId as
   integer) as integer
function pxi_CardNameToID(theFile as RADynFileFmtIO, StackObject
   as PXIObject, cardName as string) as integer
function pxi_CardNameToNumber(theFile as RADynFileFmtIO,
   StackObject as PXIObject, cardName as string) as integer
function pxi_CardNumberToID(StackObject as PXIObject, cardNumber
   as integer) as integer
function pxi_CardNumberToName(theFile as RADynFileFmtIO,
   StackObject as PXIObject, cardNumber as integer) as string
function pxi_CdNumOfBgIDfromCardID(theFile as RADynFileFmtIO,
   theStack as PXIObject, bkgndId as integer, cardId as integer)
   as integer
function pxi_CdNumOfBgIDtoCardID(theFile as RADynFileFmtIO,
   theStack as PXIObject, bkgndId as integer, cardNumber as
   integer) as integer
function pxi_CountBkgnds(StackObject as PXIObject) as integer
function pxi_CountButtons(theObject as PXIObject) as integer
function pxi_CountCards(StackObject as PXIObject) as integer
function pxi_CountFields(theObject as PXIObject) as integer
function pxi_CountParts(theObject as PXIObject) as integer
function pxi_FieldIDtoNumber(theObject as PXIObject, fieldId as
   integer) as integer
function pxi_FieldNameToNumber(theObject as PXIObject, fieldName
   as string) as integer
function pxi_FieldNameToPartID(theObject as PXIObject, fieldName
   as string) as integer
function pxi_FieldNameToPartNumber(theObject as PXIObject,
   fieldName as string) as integer
function pxi_FieldNumberToPartID(theObject as PXIObject,
   fieldNumber as integer) as integer
function pxi_FieldNumberToPartName(theObject as PXIObject,
   fieldNumber as integer) as string
function pxi_FieldNumberToPartNumber(theObject as PXIObject,
   fieldNumber as integer) as integer
function pxi_FindStackOwningBkgndID(theFile as RADynFileFmtIO,
   bkgndId as integer) as integer
function pxi_FindStackOwningBkgndName(theFile as RADynFileFmtIO,
   bkgndName as string) as integer
function pxi_FindStackOwningCardID(theFile as RADynFileFmtIO,
   cardId as integer) as integer
function pxi_FindStackOwningCardName(theFile as RADynFileFmtIO,
   cardName as string) as integer
function pxi_PartIDtoName(theObject as PXIObject, partId as
   integer) as string
function pxi_PartIDtoNumber(theObject as PXIObject, partId as
   integer) as integer
function pxi_PartNameToID(theObject as PXIObject, partName as
   string) as integer
function pxi_PartNameToNumber(theObject as PXIObject, partName as
   string) as integer
function pxi_PartNumberToID(theObject as PXIObject, partNumber as
   integer) as integer
function pxi_PartNumberToName(theObject as PXIObject, partNumber
   as integer) as string
```

With these methods, you should be able to resolve just about anything about an object.

Unique ID's

Unique ID's are strings that uniquely identify every property of every part of every object in a project file. They are of the form *Taa,bb,cc* where *T* is the type of object (C for card, B for background, S for stack, or T for report template), *aa* is the ID number of that object, *bb* is the ID number (**not** index number!) of the part (or 0 or -1 for the object itself), and *cc* is the number of the property (or -1 for the object or part itself). They may also be of the form *Gnn*, with *G* standing for the global environment and *nn* being the property number.

Unique ID's are useful when you want to get or set a property, part, or object without having to do anything with PXIObjects or StackWindows or PXIObjectIdentityLib. The scripting system, CHASM, will be using Unique ID's to get and set properties by script. All you need to work with a Unique ID is an RADynFileFmtIO and the Unique ID itself.

To generate a Unique ID, one big master method is used:

```
function pxi_MakeUniqueIDfromAnyRef(theFile as RADynFileFmtIO,
   "id"|"name"|"", <stackID>|<stackName>|<empty>,
   "card"|"background"|"stack"|"template", "id"|"name"|"number",
   <objectID>|<objectName>|<objectNumber>,
   "part"|"button"|"field"|"", "id"|"name"|"number"|"",
   <partID>|<partName>|<partNumber>|<empty>,
   <propertyName>|<propertyID>|<empty>) as string
```

This method will automatically decide which methods from PXIObjectIdentityLib to use to resolve the reference, and eventually return a Unique ID.

There are a few rules about what parameters you can pass as empty and which ones you can't:

- If you're accessing everything by ID, you don't need to pass an RADynFileFmtIO. Otherwise, you MUST pass an RADynFileFmtIO.
- You MUST pass an object.
- If you're accessing a stack or template as the object, you don't need to pass a stack.
- If you're accessing a card or background by ID, you don't need to pass a stack.
- If you're accessing a card or background by name or number, you MUST pass a stack.
- Passing a part is optional; if it isn't passed, the Unique ID will refer to a property of the object or the object itself.
- Passing a property is optional; if it isn't passed, the Unique ID will refer to the part or the object itself.

There is one other useful function that will return a Unique ID. It is used by the StackWindow class to determine what the user is clicking on:

```
function pxi_FindUIDClicked(theCard as PXIObject, theBkgnd as
   PXIObject, theStack as PXIObject, x as integer, y as integer,
   property as variant) as string
```

Any of the three objects (card, background, and stack) may be replaced with `nil` if needed.

Once in possession of a Unique ID, we may use three methods with it, which look similar to the three methods used with a PXIObject:

```
function pxi_GetObjectOrPropertyByUID(theFile as RADynFileFmtIO,
   uniqueID as string) as variant
sub pxi_SetObjectOrPropertyByUID(theFile as RADynFileFmtIO,
   uniqueID as string, value as variant)
sub pxi_DeleteObjectOrPropertyByUID(theFile as RADynFileFmtIO,
   uniqueID as string)
```

If we want to display a Unique ID to the user, it wouldn't make much sense to show it in its internal form. Here's a function that will convert it to xTalk syntax:

```
function pxi_UniqueIDtoXionString(uniqueID as string) as string
```

The resulting string is something like `"the script of part id 12 of card id 1984"`.

That's all for now.