Early ExtendScript Experiences

May 4th, 2011 by Simon Bate

FrameScript has long been our go-to tool for FrameMaker automation. When our clients have needed additional functionality in their FrameMaker environment, we have suggested that they buy FrameScript. Using it we can make writers' tasks easier, we can add automation to FrameMaker processes, or even implement unattended operation of lengthy tasks. Unfortunately, as inexpensive as FrameScript is, the added software cost has been just a bit too much for some clients.

Now in FrameMaker 10 the ExtendScript scripting environment is build into the product. ExtendScript allows automation of most FrameMaker tasks and, through the ExtendScript Toolkit, allows scripts to use other ExtendScript or ActionScript objects. If our clients were going to be asking for tools that use ExtendScript, I thought I'd take a look at it.

Before going any further, I need to "fess up." I've got a lot of experience using FrameScript and I have a good understanding of the FrameMaker document model…from the FrameScript perspective. However (confession time), I haven't spent a great deal of time programming in the FDK. Not that I haven't wanted to. As a result, I'm not as familiar with the intricacies of FDK programming as I would like to be. A bit longer with ExtendScript and I suspect that will change. If some of my scribblings here sound a bit babe-in-the-woods, so be it. On the other hand, if you, like me, are approaching ExtendScript with a FrameScript background (or with no FDK experience at all), you may find this post helpful.

Scripting language

The FrameScript language has always been somewhat quirky. The fundamental syntax of the language is similar to BASIC. Beyond the basic control structures, FrameScript allows you to access FrameMaker's object model. It then adds some really cool functionality with ESLObjects, including forms objects (which can create sophisticated dialog boxes), access to ActiveX objects, access to databases, and so on. Because FrameScript grew up with the FDK in mind, it provides constructs such as Loop ForEach, which allows you to access many common FrameMaker document objects in sequence. FrameScript also contains a number of commands that provide FrameMaker functionality directly (Copy, Cut, and Paste, for instance).

Those of us who have written FrameScripts for a while have become used to some of the "convenience" functions provided in FrameScript. They have provided shelter from the details of FDK programming.

ExtendScript is essentially JavaScript that uses additional objects that allow access to the Windows environment (the ExtendScript toolkit CS5) and the FrameMaker object model. Syntactically speaking, it's quite clean. If you've had any experience with JavaScript, the ExtendScript syntax should come naturally.

Unfortunately, the documentation provided by Adobe on ExtendScript is woefully lacking (more on that later). There have been a handful of blog posts, which introduce some very basic ideas of ExtendScript. Unfortunately, there was little in these posts that I hadn't figured out already. What was missing were some hints that I could use to begin to replicate my FrameScript experience in ExtendScript.

search this site Search

Sign up for our newsletter

Email Address*

First Name

Last Name

* = required field

Preferred Format

[✓ HTML] Text Mobile

Subscribe

Archives

Select Month

Posts by author

Alan Pringle
Gretyl Kinsey
Holly Mabry
Matt Sullivan
Ryan Fulcher
Sarah O'Keefe
ScriptoriumTech
Simon Bate

Posts by category

Conferences
Events
Humor
News
Opinion
Reviews
Tech Tips

Talk to us

info@scriptorium.com
919-433-2618
866-605-9677

What is missing or different?

The first question that came to my mind was “how can I write something to the FrameMaker console, such as a ‘Hello World.’ string, just to see that I’m doing things right?”

After some digging (and playing with the JavaScript alert() function), I found the Err() method, which allows scripts to write to the console.

```
Err('Hello World.\n');
```

Note that unlike the FrameScript write console command, the Err() method requires a newline (“\n”) at the end of each line.

Actually, Err is one of many methods associated with the Globals object. At this point, it’s worth pointing out the Object Model Viewer. To find out about any of the classes, properties, and objects in ExtendScript, go to Help > Object Model Viewer in the ExtendScript Toolkit application. Then under the word “Browser,” click the drop-down and choose “Adobe FrameMaker-10 Object Model.”

When you scroll through the various objects in ExendScript Object Model Viewer, you may not initially notice the Globals object. Look for it, because it contains a number of useful methods for interacting with the FrameMaker user interface, through the FDK. These include methods to modify the FrameMaker menus, get font information, push and pop the clipboard contents, display alerts, and print debugging information, among others.

My second big question was how to access the current FrameMaker session. In FrameScript there are two very useful properties, ActiveDoc and ActiveBook, which you use to find the FrameMaker file or book that is currently active. Quite often this is how you address the current document. Although FrameScript lets you write simply “ActiveDoc” or “ActiveBook”, these are both properties of the session object, so the full syntax should be “session.ActiveDoc” and “session.ActiveBook”. In ExtendScript, you use the app object to locate session information.

```
var vDoc = app.ActiveDoc;
Err('ActiveDoc name is: '+vDoc.Name+'\n');
```

Finding the app object in the Object Model Viewer is somewhat problematic. The app object is the only object that begins with a lowercase letter. A sorting quirk in the Object Model Viewer puts “app” at the end of the object list, AFTER all other objects (after “XRefFmt”).

Question three was how to get only the text out of a FrameMaker object (without markers, propery changes, conditions, and so on). FrameScript provides a property (**text**) that returns only the text of a paragraph. In much of my FrameScripting work I use the text property much more frequently than the Get TextList command, which returns the full content of the paragraph. Once I have the raw text I can perform many other tasks, such as search it for a substring, report it to the user, manipulate it, or use it to build a new paragraph somewhere else.

From what I can see, ExtendScript doesn’t offer anything like the text property. If you want only the text, you have to create a function like this:

```
function getText(vPgf) {
var vText = '';
var vTextItem;
var vTList = vPgf.GetText(Constants.FTI_String);
for (var i=0; i &lt; vTList.length; i++) {
vTextItem = vTList[i];
vText += vTextItem.sdata;
}
return vText;
```

Contact form

Content Strategy 101 is now available!

Available in print, EPUB, and Kindle formats. Get your copy today!

Follow Scriptorium

via RSS
on Twitter
on Facebook

```
}
```

The last of my big questions went unanswered: How do I list the properties of an object?

When writing and debugging FrameScripts, I have found the **properties** property to be extremely useful. Seeing all the properties of an object can bring an error into sharp focus.

```
write console vPara.properties;
```

The ExtendScript methods getProps() and PrintPropVals() are only marginally useful. The PrintPropVals() method does not convert the PropVal identifiers into strings, which makes the output very hard to read. At least PrintPropVals() expands the property value information and indicates the value type along with the value.

At this point, I cannot find anything in ExtendScript that is as useful as the FrameScript **properties** property. Because I typically use it during debugging, it's not worth creating a massive table to associate the identifier values with strings. If anyone has a suggestion for this shortcoming, I'd love to hear it.

Some Caveats

In my dabbling I found a few FrameScript habits that got me into trouble in ExtendScript.

Case matters

In FrameScript, case does not matter for properties, keywords, and variables. In ExtendScript, it does. Thus, to get the name of a document, vDoc.name does not work, but vDoc.Name does.

Null objects vs. empty objects

In FrameScript, you create a variable to hold an object (for example a table). If the object is null, the value of the variable is null. In ExtendScript, variables are JavaScript objects. JavaScript objects contains a number of properties in addition to the actual value. Thus, even if the object you store in the variable is null, the object variable itself is not null. A common FrameScript trick is to loop through a set of objects until the null object is encountered:

```
Set vTable = ActiveDoc.FirstTblInDoc;
Loop While(vTable)
// Do some stuff
// Get the next table
Set vTable = vTable.NextTblInDoc;
EndLoop
```

Using "While(vTable)" will not work in ExtendScript. Instead, you use the .ObjectValid() method to check if the object is valid or not (you can also check the value of the .id property).

```
var vTable = app.ActiveDoc.FirstTblInDoc;
while (vTable.ObjectValid()) {
// Do some stuff
// Get the next table
vTable = vTable.NextTblInDoc;
}
```

Documentation issues

My greatest disappointment with the Framemaker 10 ExtendScript implementation is with the documentation. I can work around just about all the other differences I've described thus far (even the lack of the **properties** property), but the lack of documentation is truly unfortunate.

The ExtendScript documentation itself predates FrameMaker 10, and thus has nothing about working in the FrameMaker object model. That is understandable. But it would have been good to have a FrameMaker supplement to the ExtendScript documentation.

When I first opened the ExtendScript Toolkit and found the Object Model Viewer I thought I had found what I was looking for: descriptions of the FrameMaker 10 classes, properties, and methods.

To my great disappointment and annoyance, the definitions contain no text; they list the FrameMaker 10 objects and their relationships to other classes, properties, and methods. But there is NO (none, bupkis, zilch) documentation about what each class, property, or method does. I was able to recognize many objects from my familiarity with the FrameMaker object model. But many others remained fairly obscure. For these, I was able to use the FDK reference documentation to determine what these objects do. The FDK is not part of the downloaded ExtendScript package, but is essential nonetheless. The omission of any descriptive text from the object model data is *serious* shortcoming, which I hope Adobe will remedy SOON (please?).

Note that this is a FrameMaker-specific issue: the Object Model Viewer has quite complete documentation for the JavaScript and ScriptUI classes.

In summary…

ExtendScript has the potential to be a useful addition to FrameMaker. Although the cost of upgrading to FrameMaker 10 must be kept in mind, once you have upgraded, the only additional cost for creating scripts will be the time spent developing, testing, and debugging. However, given it's immaturity and poor documentation this could be significant. On the other hand, FrameScript is a mature product with many convenience functions, extended capabilities, usable (and useful) documentation, and an experienced base of script developers. The lower cost of FrameScript development could easily make up for the cost of FrameScript licenses.

Have you experimented with ExtendScript in FrameMaker 10? What has your experience been?

Tags: ExtendScript, FrameMaker, FrameScript, scripting
Category: Reviews, Tech Tips, Tools
Permalink | 7 Comments »

Tweet 8

« Previous The devolution of DITA editors Webcast: The State of Structure, 2011 Next »

7 Responses to "Early ExtendScript Experiences"

1. *Michael Müller-Hillebrand* says:
May 5, 2011 at 4:39 am

As coming from the same background I have the same experiences! At the moment if the task is not an easy one, it is safer to offer FrameScript programming.

Nevertheless I have a commercial project finished which involved processing all files of a book, finding structured elements, adding an element and an anchored frame with some content. In the end it is pretty clear, but the lack of documentation and examples make it somewhat time consuming.

BTW, you can use Console() to output something without adding your own \n.

Reply

2. *Rick Quatro* says:
May 5, 2011 at 11:51 am

I have a similar background as Simon and Michael; I started using FrameScript (in 1998!) with no FDK experience. However, I soon found the FDK documentation to be an essential learning tool. It helped me understand the FrameMaker object model and gave code samples that I could "translate" into FrameScript's syntax.

You will definitely need the FDK docs to use ExtendScript. Adobe's lack of documentation and examples is pitiful and is inexcusable for a company of their stature. The FDK docs still have FrameMaker 7.0 on the covers, and the later version supplements lack useful examples. But, as of right now, that is the best you are going to get for documentation.

As you might guess, I am still very bullish on FrameScript.

Reply

3. *Simon Bate* says:
May 5, 2011 at 12:56 pm

Thanks for the tip, Michael!

I agree with Rick on the FDK and FrameMaker scripting. When I was first learning FrameScript I quickly discovered that the FDK documentation provided conceptual information that was beyond the scope of the FrameScript docs. (Rick also gets a great deal of credit for documenting and publishing what he learned.)

Reply

4. *Peter Gold* says:
May 6, 2011 at 12:10 pm

I often mention FrameMaker on the InDesign forum, usually because FrameMaker has a feature that InDesign lacks, or because a FrameMaker feature is better or does more or is implemented more smoothly than InDesign, but sometimes NOT! After all, FrameMaker is only human.

So now, I'm looking from the other end of the telescope for a bit. When I first taught InDesign 1.x to a group at a newspaper publisher, they were looking into it as a personal-computer solution to the dedicated and proprietary minicomputer systems they'd been using. They'd customized the tools for their workflow, by a mix of commercial add-ons, custom-third-party development, and in-house development. I was sure that I learned more from the questions that illuminated their needs, than I was able to offer as solutions. The most shocking to me as a FrameMaker trainer and user were the many variations on working with individual pages – something like the disconnected-pages nightmares we try to avoid in FrameMaker as much as possible. Another issue was asset management across multiple workers, workstations, publications, and deadlines, and almost everything down to the smallest component was an asset to be managed into and out of databases. When I suggested they check into the Adobe InDesign third-party developer's forums, where the product developers are often participants, they said they already did, and "often they ask us if we'd solved problem X or Y, or if we had suggestions on how to!"

InDesign has had ExtendScript for some time. My hands-on experience with it is as near to nothing now, as before it was introduced, but on InDesign fora, I've scanned over ESTK postings – questions, solutions, examples, etc. – and have used scripts that users and developers have generously provided. The ability to be scripted is one of InDesign's strongest features, IMO. Because it's so readily available to those who can actually write or modify scripts, the unofficial InDesign development team far outnumbers the official one, and the speed with which new, often niche-focused, "features" are added is almost instantaneous, compared to waiting for new releases with the hope that a prayer will have been answered.

Of course the application object models are completely different, but when I was learning to program dBase II the "salmon" way – swimming upstream until I got to my goal – a friend said "if you know how to program in any language, you've got most of the others in your sights." I don't know if InDesign's ESTK documentation is any better than FrameMaker's, but it might be worth searching Google for "InDesign Extendscript scripts examples documentation tutorials," and similar terms, without quotes, to see if there's anything that might contribute to winning the ESTK battle in FrameMaker.

Although the long-standing FrameScript solution won't be the only game in town, developers like Rick who migrate their hard-earned FrameMaker-customization expertise to ESTK will still have a lot of new solutions and products to offer as FrameMaker continues to evolve.

For example, currently, the only commercial FrameMaker-to-InDesign conversion tool is the DTP Tools MIF Filter plug-in for InDesign. It's actually free; you buy credits, like phone-card minutes, to convert FrameMaker content to InDesign yourself, or you buy their conversion services. The name makes it clear that it works with MIF, not directly with FrameMaker binary files, and that it's a filter – a map between FrameMaker features and constructs and InDesign's counterparts, where they exist, and where they don't, there's some fancy footwork that often works acceptably, but sometimes steps foot on a banana peel. I'd expect that it's possible to create true conversion utility with Extendscript; not easy, but possible. I'm not pointing this out as a prelude or encouragement for smooth migration to InDesign that accelerates the abandonment of FrameMaker. Rather, I think there's a need that some users have, and there's value in filling it; even more radical is the idea that roundtripping might have value for some other groups of users.

Reply

5. *Simon Bate* says:
May 6, 2011 at 1:11 pm

Hi Peter, I probably should have pointed out that ExtendScript has been around for a while, so practical matters of programming in ExtendScript (creating GUIs, complex data storage issues, and the like) have been addressed in various user groups. That I have no problem with.

My main beef is that the FrameMaker-specific object reference in the Object Model Viewer is void. What does AllocatePropVals() in Globals do? It doesn't tell you. How do you use Doc.AddNewBuildExpr()? Nothing. You can go to the FDK to find out, but that sort of obviates the need for an Object Model Viewer.

Reply

6. *Writer* says:
May 12, 2011 at 4:05 am

Hi Simon,

Thanks for you feedback.

We are working on the documentation and will address that shortcoming, in a very short while.

Thanks.

Reply

7. *Pierre-Louis Desrosiers* says:
June 25, 2012 at 4:09 pm

Hi,

Thanks: Your comments are by far the most substantial I have found yet on ExtendScript (in close relation to FrameMaker) anywhere.

As for others commenting on the FDK as essential reference material, I agree, but it does not (seem to) help me when I need to access the ExtendScript environment's (i.e. not FrameMaker's) facilities.

(A current bugaboo of mine: trying to use a list selection to control script flow in ExtendScript! It feels unnecessarily opaque—because of poor documentation I suspect. To note: since FrameMaker 9, I similarly experienced problems whenever I tried to "customize" the toolbars, etc., which used to be a cinch for me in FrameMaker 5 to 7—and, I suspect, similarly because of poor documentation.)

Framescript is very convenient for rapid development, so much so that it always feels well worth it. But as it is not free, it cannot be deployed on just any FM target.
The FDK is solid, free (when running MS VStudio Express), and can be readily be deployed on (almost) any FM target. But it can be a pain (!) to code…
If ExtendScript was/became more convenient as a "port" target (for wich it lacks currently), it could become a superb deployment tool/medium for any FrameMaker 10 target.

Regards.

PLD

Reply

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Submit Comment