On Code Generation

Posted At : January 17, 2007 6:00 AM | Posted By : Steve
Related Categories: Productivity, ColdFusion

I just read Brian Rinaldi's article on **building a code generator** . He lists five steps to build a code generator. I think the article is a good one overall, but it inspired me to point out where I would (and have) done things differently.

His steps are as follows (quoted directly from is article):

1. the Admin API in ColdFusion MX 7;
2. accessing database metadata in varying RDMS;
3. converting this data to a basic XML format;
4. transforming your XML into code with XSL;
5. the CFML code necessary to bring the first four parts together into a functional application.

1. The Admin API in ColdFusion MX 7

Brian uses this to get a list of available databases. I agree that this is nice, but I don't think it is an essential part of code generation. If someone has to type in the datasource, that doesn't completely negate the value of the code generator.

If you choose to use the Admin API, however, you might consider running your code as an extension in the ColdFusion Administrator. If you run your code in the CF Admin and you use the Application.cfm at the root of the administrator (in CFMX 7), then you will have a variable called "variables.factory". This holds the Admin API - and without you having to know the username and password. I don't think that this is a security problem as someone had to enter to the username and password to get your code to run in the first place.

I will also add in that you can see an example of how to do this as well as how to get a list of supported datasources can be found in my **CodeCop** code checker program.

2. Accessing database metadata in varying RDMS

I agree with Brian that this is an essential step (though **Peter Bell** has some good thoughts on code generation and the limits of database metadata). The approach that he used, however, means that he has to write different code for each database. He could have gotten the same information by using **DataMgr**. A call to DataMgr's **getDBTableStruct()** would get all of the same information for any database supported by DataMgr (which currently includes MS Access, MS SQL Server, MySQL, and PostGreSQL).

3. Converting this data to a basic XML format

This step isn't necessary unless you want to use XSLT for your code generation (see **Peter Bell's thoughts on XSLT code generation**). Assuming that you want to use XSLT, however, DataMgr can help you out again. In fact, you can skip the previous step and just have DataMgr give you an XML representation of the table structure. A simple call to getXml() will return an XML string for that table. If you need to add more information to the XML, just use ColdFusion's built in XML functions to add data to the XML.

4. Transforming your XML into code with XSL
and
5. The CFML code necessary to bring the first four parts together into a functional application

These two steps are really the same for me. This is the "generate the code" portion of the process. No doubt essential (it is the point, after all). I have already mentioned that XSLT may or may not be the best choice

Let me just mention one other tool that may be of use if you decide not to use XSLT (which is a fine option).

I have a small component (**tag.cfc**) which can be extended to be an object representation of virtually any tag. I have already extended it for cfcomponent, cffunction, and cfargument (as well as some of my own custom tags). You can use it to generate code for virtually any tag. I don't want to imply that it is better than XSLT, just another approach.

Conclusion

I really liked Brian's article. It got me thinking about how I do things differently. I have been intending to work on my code generator again for some time (I actually use it for myself on most new projects, but it isn't quite "ready for Prime Time" yet). Hopefully I can work on that again as soon as I release DataMgr 2.0.

UPDATE

Here is a handy link to Brian's response (so you don't have to copy and paste from the comment below):

http://www.remotesynthesis.com/blog/index.cfm/2007/1/17/Generating-Code-vs-Generating-Applications