Code generation in re-motion

re-motion makes good use of program generation
and comes with three command-line program generators:

dbschema.exe	generates a database schema for persisting domain object instances
uigen.exe	generates a ASP.NET-application from domain object classes
wxegen.exe	generates boilerplate code for re-call page-functions from re-call headers
run-time code generation	transparently sub-classes domain object classes for instantiation

Program generation happens also at run-time. Code for domain object classes is generated for binding and persistence from declared base-classes. What's more, some parts of re-motion organize code in re-mix for better separation of concerns.

Don't confuse these program generators. They corresponding purposes don't overlap. With the exception of *run-time code generation*, all program generators are command line tools, but `wxegen.exe` is not usually invoked by the programmer. (It is invoked by Visual Studio during a build.) The use of each program generator is discussed in the PhoneBook tutorial, and also in this wiki.

An entirely different type of program generation is the creation of so-called "generated types" at run-time. Nothing must be invoked, because this program generation works tacitly in the background and is completely transparent to the programmer.

`dbschema.exe`

`dbschema.exe` is always the first program generator used, because it derives the database schema for persisting domain objects. `dbschema.exe` inspects domain object assemblies to find domain object classes, their properties and attributes and generates the database schema – tables and views. d`bschema.exe` can be invoked many times in the course of a project, because the programmer does not modify the generated database schema. (This is in contrast to `uigen.exe` that can be run only once before making changes. You must start from scratch every time you run `uigen.exe`.)

- more on `dbschema.exe`: dbschema.exe
- using dbschema.exe

`uigen.exe`

`uigen.exe` enters the stage as soon as you not only have the properties and attributes for your domain object classes, but also query methods, factory methods and behavior – in short: the "domain". You are ready for the web application.

`uigen.exe` works like `dbschema.exe` in that it inspects the domain object classes, properties and attributes in domain object assemblies `uigen.exe`'s use is more delicate, however, because it generates the starting point for programming your web application. You customize and extend that web application by modifying the generated files. `uigen.exe` is not a round-trip tool. If you change properties (types, constraints) in one or more domain object class, you must `uigen.exe` again and start over with your modifications.

- more on `uigen.exe`: uigen.exe
- using uigen.exe

Both `uigen.exe` and `dbschema.exe` inspect domain object assemblies, but...

`uigen.exe` is more picky than `dbschema.exe` when it comes to usage. Both program generators must load domain object assembly(s) for discovering domain object classes, but they resort to different libraries for doing so.

- `uigen.exe` (an old program) uses an improvised an fragile mechanism
- `dbschema.exe` (a new program) uses a mature and robust mechanism

In practice, you won't have problems with `dbschema.exe`, because it usually does the right thing. `uigen.exe` is a robust program that has been in use for years, but that's exactly the problem: it is dated. Don't draw conclusions from `dbschema.exe` usage to `uigen.exe` usage, and the other way around. How to avoid `uigen.exe` assembly discovery failures is explained in detail in using uigen.exe.

`wxegen.exe`

`wxegen.exe` will not be called by you, at least not usually.
`wxegen.exe` works transparently in the background in Visual Studio.
`wxegen.exe` parses source code, i.e. it does not inspect assemblies.
`wxegen.exe` is only interested in so-called "WXE headers" – XML written into comments for pages derived from `EditFormPage`.

An `EditFormPage` represents a web edit form for data entry, but it can be "called" with parameters and has a return value – like a function. Generating the boilerplate for passing parameters and returning a value (if desired) is what `wxegen.exe` is designed to do for you.

Whenever you change something in a source file constituting a class derived from `EditFormPage`, `wxegen.exe` will run and re-generate the boilerplate. This brief explanation probably remains somewhat mysterious unless you know what re-call is. There is a known problem with `wxegen.exe`: it does not work the first time, due to a Visual Studio bug. This is explained here: re-motion mascot bug -- wxegen.exe does not work.

Run-time code generation

re-store is re-motion's persistence layer, i.e. it provides O/R-mapping for domain object classes, as declared by the programmer. Persisting such objects requires sophisticated machinery that can't be provided by the common baseclass `DomainObject` alone. For this reason, *mixin* technology is used to "mix in" the desired behavior at run-time, before instantiating a given object for the first time. Mixins in general are explained in wikipedia.

You can find an article series about re-motion mixins in the re-motion team-blog and in the exhaustive documentation here: re-mix. The hows and whys of generated types can be found here: generating domain object types at run-time.

See also

Each program generator is discussed in detail in the PhoneBook tutorial. For reference (parameters, errors) use the corresponding wiki-pages:
dbschema.exe
uigen.exe
wxegen.exe
generating domain object types at run-time