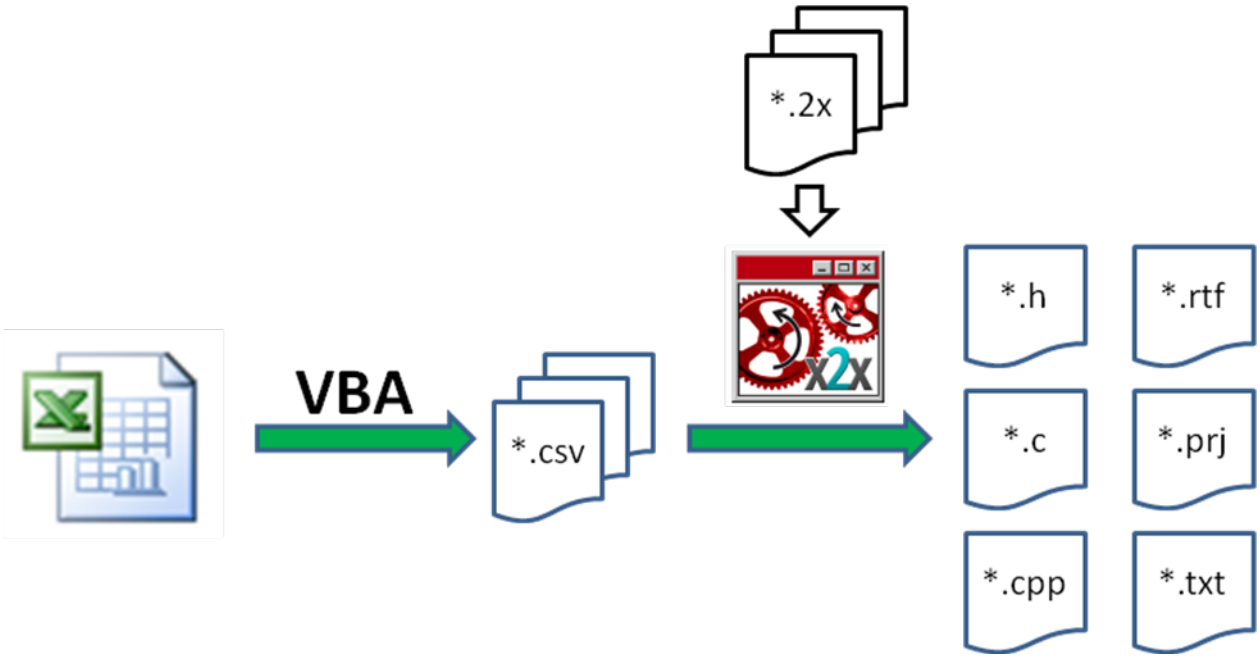
Sourcecode generieren? Ja, aber wie?

Vor der Aufgabe Sourcecode zu generieren steht man zum Beispiel immer dann, wenn Variationen eines bestehenden Sourcecode erstellt werden müssen. Abhängig davon, wie viele Variationen erstellt werden müssen und in wie weit sich diese voneinander unterscheiden, lohnt sich der Einsatz eines Sourcecode-Generators.



Beispiel: Generierung von Sourcecode aus einem Excel-Sheet

1. Ohne X2X?

(Lesen Sie in Kapitel 2. weiter, wenn Sie sich bereits für X2X entschieden haben!)

Die einfachste Art von Sourcecode-Generator nimmt in einer Sourcecode-Textschablone lediglich Textersetzungen vor. Ein typisches Beispiel dafür ist der C/C++ Präprozessor. Mit der Anweisung #define XXX YYY, wird der C/C++ Präprozessor angewiesen nachfolgend den Text XXX immer durch YYY zu ersetzen. Typischer Weise werden dabei Platzhalternamen durch konkrete Werte ersetzt.

Der C/C++ Präprozessor ist sogar noch in der Lage, abhängig von Platzhalterwerten Bedingungen auszuwerten und abhängig vom Ergebnis, ein Stück Sourcecode ein- oder auszublenden.

Obwohl damit bereits die Grenzen des C/C++ Präprozessors erreicht sind, lässt sich mit den beiden Paradigmen „Platzhalter ersetzen“ und „Bedingung auswerten“ ein eingeschränktes Variantenmanagement installieren. Allerdings kann die Verwaltung dieses Variantenmanagements schnell zur Herausforderung werden, da oft keine Zusatzdokumentation darüber angelegt wird, welche Werte ein Platzhalter annehmen darf und wie sich die Änderung eines Platzhalterwertes auswirkt.

Das Haupthindernis bei der Verwendung der C/C++ Präprozessors als Variantengenerators ist allerdings, dass er keine Listen verarbeiten kann. Zum Beispiel ist es nicht möglich für eine Liste von "Enumeration" den Sourcecode für eine Switch-Case-Anweisung zu generieren. So gesehen sind die Möglichkeiten mit dem der C/C++ Präprozessors Sourcecode zu generieren sehr, sehr beschränkt.

Natürlich kann man dieses Problem umgehen, indem man den Sourcecode-Generator mit einer Hochsprache wie zum Beispiel C, C++ oder Java schreibt, da jede Hochsprache die Möglichkeit bietet Listen über Schleifen zu verarbeiten. Aber leider, leider wird es dann kompliziert. So kompliziert, dass meist lieber von der Verwendung eines Sourcecode-Generators wieder Abstand genommen wird.

Verwendet man eine Hochsprachewie z.B. C, C++ oder Java zum Implementieren eines Sourcecode-Generators, so muss die Ausgabe über zahllose nicht zusammenhängende PRINT-Ausgaben erstellt werden, wodurch ganz schnell der Überblick verloren geht. Andererseits muss der zu erstellende Sourcecode-Generator mit Input „gefüttert“ werden können, damit er „weiß“ welchen Sourcecode er generieren soll. Dies impliziert automatisch, dass auch ein Parser implementiert werden muss, mit dem der Input eingelesen und analysiert werden kann. Den Sourcecode für einen Parser kann man zwar generieren, aber dazu muss man wiederum die Sprache verstehen, mit der ein Parser spezifiziert wird. Und schlimmer noch, man muss den generierten Parser auch in den eigenen Sourcecode-Generator integrieren, welches im allgemeinen nur möglich ist, wenn dieser einem ganz bestimmten, vom Parser-Generator vorgegebenen Designmuster folgt.

Möchte man da nicht verzweifeln? Man bräuchte doch nur so was Einfaches wie die Platzhalter Verarbeitung eines C/C++-Präprozessor, aber zusätzlich mit der Möglichkeit zur Listenverarbeitung. Und einen Parser schreiben oder auch nur integrieren – Nein danke!

Die Lösung heißt X2X! Der zur generierende Sourcecode kann mit X2X, wie bei einem Präprozessor, über Textmuster vorgegeben werden, in dem es Platzhalter gibt, die über Bedingungsanweisungen ein oder ausgeblendet und zur Listenverarbeitung wiederholt ausgegeben werden können. Aber mehr noch! Die Werte die die Platzhalter annehmen sollen, können nach eigenen Vorstellungen vorgegeben oder aus selbst definierten Rahmenparametern berechnet werden. Kurzum, mit X2X können Sie selber festlegen, über welche Parameter Sie Ihre Sourcecode-Varianten konfigurieren und in welcher Art und Weise Sie diese vorgeben wollen, (z.B. mit einer Excel-Tabelle). Genau so einfach , wie Sie mit einen X2X- Textmuster den Output, also den zu generierenden Sourcecode festlegen, genau so einfach legen Sie über einen X2X-Typen die Art und Weise fest wie Sie die Input-Parameter spezifizieren wollen.

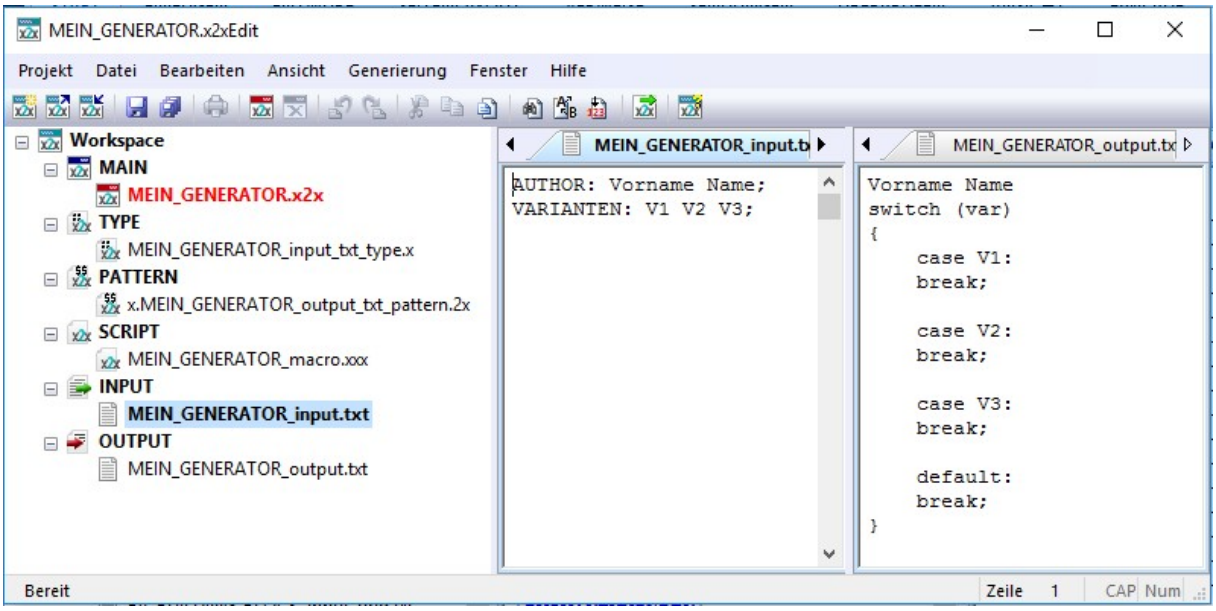
Also ist X2X ein Generator-Generator? Oder ein erweiterter Parser-Generator? Nein! X2X ist weder das Eine, noch das Andere. Er wird kein Generator generiert, es wird auch kein Parser generiert und Sie müssen auch gar nichts integrieren. X2X ist die Sprache, mit der der x2x Software-Roboter angewiesen wird auf Knopfdruck eine bestimmte Aufgabe vollautomatisch zu erledigen. Und zwar genau so, wie Sie es durch Ihre Vorgaben bestimmt haben.

Der x2x Software-Roboter arbeitet nach einem völlig neuen, aber intuitiv eingängigen Verfahren, das man am leichtesten verstehen und kennenlernt, in dem man ihn einfach anwendet. Die Test-Version des x2x Software-Roboters finden Sie unter http://x2x.sss.de zum herunterladen. Dort finden Sie

auch Kurzfilme als Einführung und Demonstrationsbeispiele, die Sie kurzer Hand und Schritt für Schritt den x2x Software-Roboter zur Generierung von eigenem Sourcecode instruieren können.

2. Mit X2X

Wenn Sie noch heute eine Sourcecode-Generator erstellen wollen, sollten zwei Voraussetzungen erfüllt sein: Erstens sollte auf Ihrem Rechner der x2x Software-Roboter installiert sein (kostenlose Testversion von http://x2x.sss.de herunterladen und installieren) und zweitens benötigen Sie einen Beispielcode, den der Generator generieren soll. Für den Anfang reicht eine einzige Datei daraus. Schon kann es losgehen.



Starten Sie die Entwicklungsumgebung x2xEdit und erzeugen Sie ein neues Projekt. (Auf der Startseite von x2xEdit wird ihnen erklärt, wie das geht.) Wenn Ihr Projekt z.B. MEIN_GENERATOR heißt, so gibt es ein Pattern mit dem Namen "*x.MEIN_GENERATOR_output_txt_pattern.2x*". Dieses öffnen und den zu generierenden Sourcecode zwischen die inneren `$BEGIN PATTERN...$` und `$END PATTERN...$` Kommandos kopieren, so dass der dort vorhandene Text ersetzt wird. Wird jetzt der x2x Software-Roboter mit F5 gestartet, so ist in der OUTPUT-Datei "*MEIN_GENERATOR_output.txt*" der zu generierende Sourcecode zu finden. Im MAIN-Skript kann man den Namen "*MEIN_GENERATOR_output.txt*" durch den richtigen Namen der Sourcecode-Datei ersetzen. Damit ist schon der Grundstein für den Sourcecode-Generator gelegt.

Als nächstes werden sukzessive alle variablen Bestandteile des Sourcecode im Pattern ersetzt. Im einfachsten Fall sind dies gewöhnliche Platzhalter. Der konkrete Wert eines Platzhalters wird in der INPUT-Datei "*MEIN_GENERATOR_input.txt*" abgelegt. Damit der Generator den Wert eindeutig dem zugehörigen Platzhalter zuordnen kann, muss in der TYPE-Datei "*MEIN_GENERATOR_input_txt_type.x*" die Struktur der INPUT-Datei festgelegt werden. Diese wird dann dazu verwendet die Werte aus der INPUT-Datei zu extrahieren.

Eine gängige Art und Weise ist es, die Struktur der INPUT-Datei ist es diese zeilenorientiert aufzubauen und dem Wert eines Platzhalters gerade den Namen des Platzhalters als Schlüsselwort voranzustellen.

So würde zum Beispiel einem Platzhalter `$:AUTHOR$` aus der PATTERN-Datei, die Zeile
`AUTHOR: Vorname Name;.`
in der INPUT-Datei den Wert "`Vorname Name`" zuordnen, wenn die Struktur in der TYPE-Datei mit
`'AUTHOR:' AUTHOR::STREAM{ ';' }`
angegeben wird.

Um dies zu testen, ersetzen Sie jetzt den Inhalt der INPUT-Datei durch obige Zeile und fügen den Platzhalter in die PATTERN-Datei ein. In der TYPE-Datei ersetzt die angegebene Strukturanweisung den Text zwischen den geschweiften Klammern von `::SEQUENCE{...}`. (Achten Sie darauf, dass die Reihenfolge der Platzhalter in der INPUT-Datei und in der TYPE-Datei übereinstimmen muss.) Wenn Sie danach den x2x Software-Roboter mit F5 starten, wird der Wert "`Vorname Name`" des Platzhalters `$:AUTHOR$` in die OUTPUT-Datei kopiert, egal wie Sie diesen Wert setzen. Der zweite Schritt zum Sourcecode-Generator besteht also darin für alle Platzhalter in der TYPE-Datei einen Namen festlegen, seien Wert mit entsprechendem Schlüsselwort in die INPUT-Datei aufzunehmen und in der PATTERN-Datei an allen Stellen, an denen er auftritt, durch den Platzhalter zu ersetzen.

Besonders effizient wirkt sich ein Sourcecode-Generator aber erst aus, wenn Listen zum Einsatz kommen, d.h. wenn sich im zu generierenden Sourcecode einzelne Fragmente wiederholen. Dies wäre zum Beispiel der Fall, wenn ein Enumeration deklariert wird und eine Switch-Case-Anweisung zu jedem Wert des Enumeration kodiert werden muss.

Dazu wird in der TYPE-Datei die Strukturanweisung `::LIST{...}` verwendet. So würde zum Beispiel in der TYPE-Datei die Zeile
`'VARIANTEN:' VARIANTE::LIST{ NAME::IDENTIFIER} ';'`
für die INPUT-Datei eine Liste von Bezeichner spezifizieren, wie zum Beispiel
`VARIANTEN: V1 V2 V3;.`
In der PATTERN-Datei kann dann über die Schleifen-Anweisungen `$BEGIN ALL ...$ und $END ALL ...$` ein entsprechendes Code-Fragment wiederholt ausgegeben werden. So würde z.B.

```
$BEGIN ALL :VARIANTE$
    case $:VARIANTE:NAME$:
    break;
$END ALL :VARIANTE$
```

zu folgender Ausgabe führen:

```
    case V1:
    break;
    case V2:
    break;
    case V3:
    break;
```

Nimmt man noch die Bedingungsanweisungen `$BEGIN IF ...$ und $END IF ...$` hinzu, so stehen einem bereits die wichtigsten Mittel zur Verfügung, die notwendig sind um den Sourcecode-Generator zu konfigurieren. Werden wie oben demonstriert, Schritt für Schritt in der PATTERN-Datei die variablen Anteile durch Eingaben aus der INPUT-Datei ersetzt, so gelangt man schließlich nicht nur zum fertigen Sourcecode-Generator, sondern erhält auch noch ganz nebenbei ein Modell, das

den Output beschreibt, denn die TYPE-Datei beschreibt nichts anderes als die Struktur dieses Modells und die INPUT-Datei definiert nichts anderes als genau eine Instanz dieses Modells.

Damit sind Sie bei einer neuen und sehr effizienten Methode der Softwareentwicklung angelangt - nämlich der "modellbasierten Softwareentwicklung". Dies ermöglicht Ihnen die Software aus einer ganz neuen Perspektive zu betrachten, viel schneller den Überblick haben und per (Generator-) Knopfdruck just-in time eine Variante des Sourcecode zu erstellen.

Auch bei der Entwicklung eines Modells kann man Verbesserungen vornehmen, in dem man sich an Paradigmen orientiert, die einem Modell vom Thema her innewohnen. Zum Beispiel kann man Tabellen natürlich gut mit einem Tabellenkalkulationsprogramm bearbeiten. Kommen also in einem Modell besonders viele Tabellen vor, so ist es naheliegend die Struktur des Modells so zu wählen, dass es mit einem Tabellenkalkulationsprogramm erfasst bzw. bearbeitet werden kann. Oder stehen zum Beispiel bei dem Modell graphische Aspekte im Vordergrund (wie z.B. bei UML), so ist es sinnvoll das Modell mit einem graphischen Editor zu erstellen. Entfernt sich dabei die Darstellung des Modells von der rein textuellen Darstellung des zu generierenden Sourcecode zu weit, werden einfach Zwischenmodelle und Modelltransformationen verwendet um alle Vorgänge weiterhin so einfach wie möglich zu gestalten.