**Erweiterte Backus-Naur Form**

Die erweiterte Backus-Naur-Form (EBNF) ist eine Art „Meta-Syntax“, also ein „Sprache“, die, wie in den diversen Programmiersprachen auch, Algorithmen und IT-Konzepte kurz und bündig – und vor Allem eindeutig – beschreiben kann.  
Die EBNF ist also verwandt mit dem Konzept des „Pseudocodes“, wo man Algorithmen in einer allgemeinen „Programmiersprache“ darstellt, ohne sich auf eine spezielle Sprache wie z.B. C#, Python, etc. festlegen zu müssen.

Beispiel:  
ZifferAusserNull = „1“ | „2“ | „3“ | „4“ | „5“ | „6“ | „7“ |   
„8“ | „9“;

Ziffer = „0“ | ZifferAusserNull;

Mit dieser Definition sind die Ziffern im Dezimalsystem eindeutig definiert. Der senkrechte Strich (Pipe-Symbol „|“) steht hierbei für die „Alternativ-Auswahl“, d.h. Entweder-Oder…

Somit ist das Beispiel so zu lesen: „Als Ziffern gelten: Entweder „1“ oder „2“ oder …“  
Diese Ausdrücke sind hier, wie in vielen IT-Umgebungen auch, mit einem Semikolon („;“) abzuschließen. Außerdem können, wie oben zu sehen, die Definitionen wie in der IT üblich in neuen Definitionen weiter verwendet werden.

Weitere Beispiele:

Zwoelf = „1“, „2“;

Zweihundertundeins = „2“, „0“, „1“;

Dreihundertzwoelf = „3“, Zwoelf;

ZwoelfTausendzweihundertundeins = Zwoelf, Zweihundertundeins;

Ohne das oben definierte Pipe-Symbol (Entweder/Oder…) kann man, durch Kommata getrennt, die Symbole einfach aneinander reihen, wie in den obigen Beispielen gezeigt.

An dieser Stelle kann man zusammenfassend schon sagen: Mit der EBNF kann man sogenannte „Terminal-Zeichen“ (im gröbsten Sinne: geläufige, darstellbare Zeichen) und algorithmische Konzepte durch Nicht-Terminal-Zeichen (z.B. ()[] | ;) definieren. Whitespace (Leerzeichen, Tabulator, Newline etc.) spielen keine Rolle – es sei denn, sie sind auch durch Anführungszeichen eingeschlossen.

Sollen die oben genannten Nicht-Terminal-Zeichen im Text auftauchen, muss man sie ebenfalls in (ggf. unterschiedliche) Anführungszeichen einschließen bzw. eine sogenannte Escape-Sequenz verwenden (z.B. stellt „\““ ein einzelnes, doppeltes Anführungszeichen dar. Dieses musste mit einem Backslash (\) „escaped“ werden!).

Weitere Konzepte als Beispiele:

NatuerlicheZahl = ZifferAusserNull, { Ziffer };

Hier werden die Natürlichen Zahlen in der EBNF definiert (Natürliche Zahlen: Alle positiven Integer außer der 0). Dabei taucht als neues Konzept die geschweifte Klammer ({}) auf.

Dieses Konstrukt bedeutet: Alles in den geschweiften Klammern kann sich beliebig oft wiederholen.  
Wichtig dabei ist: Die Klammer kann sich auch 0-mal wiederholen, also gar nicht vorkommen. Obiges Statement ist also so zu lesen:  
„Verwende eine Ziffer von 1 bis 9, dann, je nachdem, verwende beliebig viele (auch: keine!) weitere Ziffern von 0 bis 9. Damit lassen sich alle natürlichen Zahlen wie z.B. 257, 33678, 2324, darstellen.

Weiteres Beispiel:

GanzeZahl = „0“ | [„-„], NatuerlicheZahl;

Nun werden die Ganzen Zahlen definiert (positive und negative Integer inkl. 0). Hier sind die Neuheit die Eckigen Klammern ([]). Diese bezeichnen einen Teil, der (einmal) vorkommen kann, aber nicht muss. Obiges Bsp. Liest sich also wie:  
„Verwende entweder die Null oder verwende ein optionales Minuszeichen (für die negativen Zahlen), gefolgt von den weiter oben definierten natürlichen Zahlen.

Nächstes Bsp.:

LeerzeichenAlsTab = 4 \* „ “, …;

Man kann in der EBNF auch grundlegende Arithmetik betreiben. Im Beispiel sollen anstelle eines Tabulator-Zeichens 4 Leerzeichen verwendet werden. In der Syntax gibt man dann „4 \*“ und dann das Leerzeichen, „ “, gefolgt von evtl. weiterem Text, an.

Zusammenfassend: Symbole in der EBNF und deren Verwendung:

Zeichen Verwendung  
„Gleichheitszeichen“ Definition  
„,“ (Komma) Aufzählung  
„;“ (Semikolon) Ende-Zeichen  
„|“ (Pipe) Alternative  
[…] Optionaler Teil  
{…} Wiederholung (auch 0-mal)  
(…) Gruppieren  
„…“ Anführungszeichen (1. Variante)  
‚…‘ Anführungszeichen (2. Variante)  
(\*…\*) Kommentar