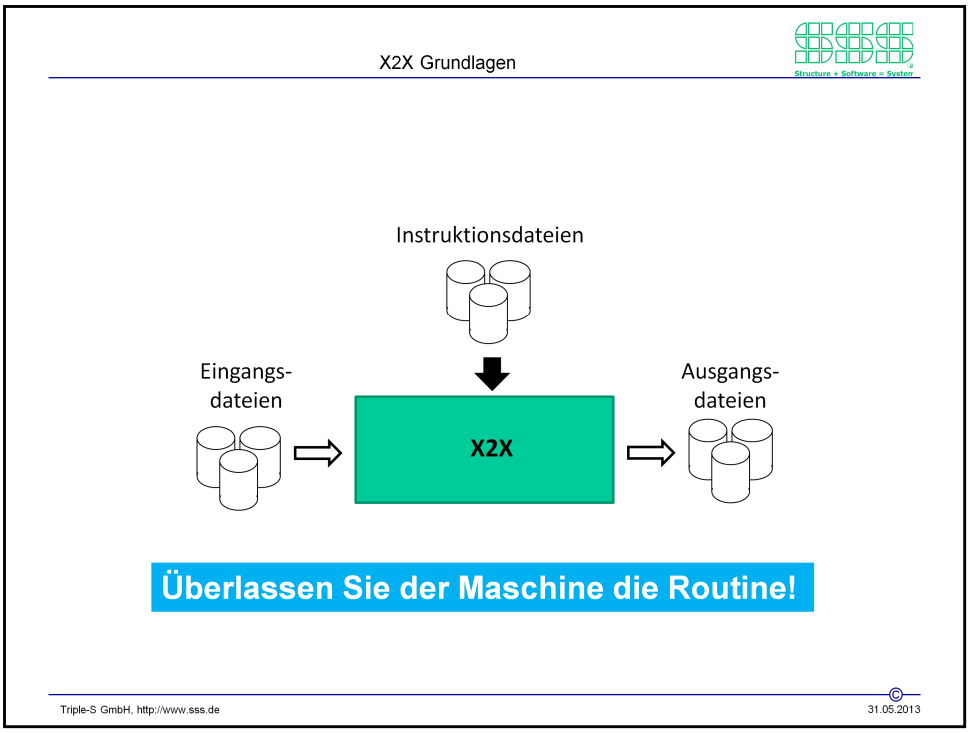


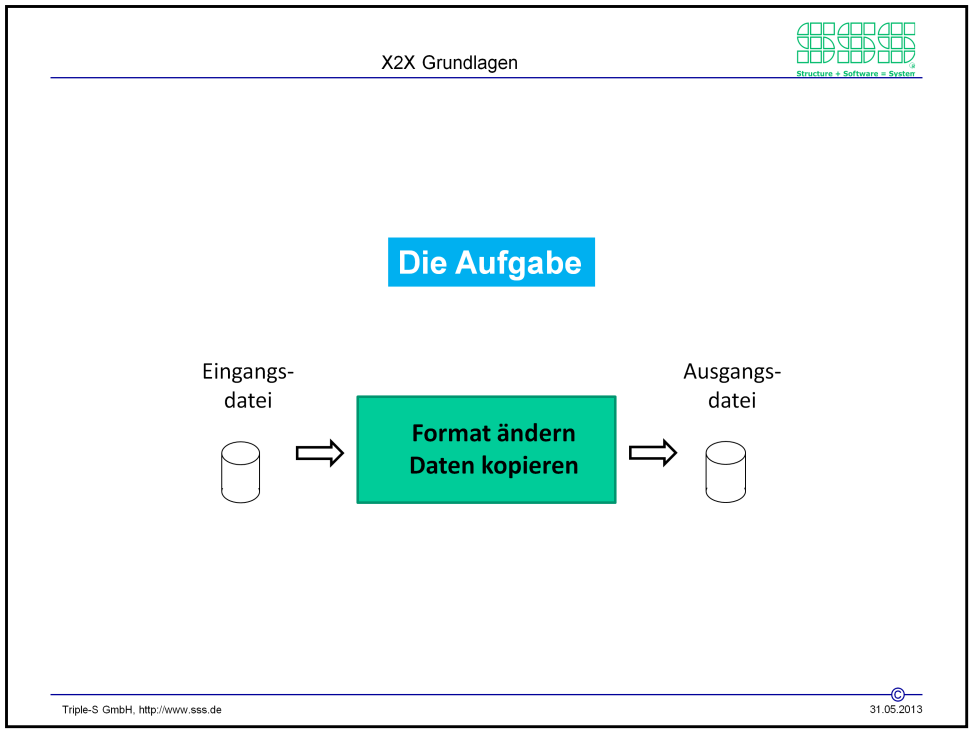
Was ist X2X?

Mit X2X kann man aus vorhandenen Dateien Daten gezielt auslesen und neue Dateien erzeugen, in denen diese Daten direkt oder in verarbeiteter Form eingefügt werden.

Dazu muss X2X über spezielle Dateien instruiert werden.

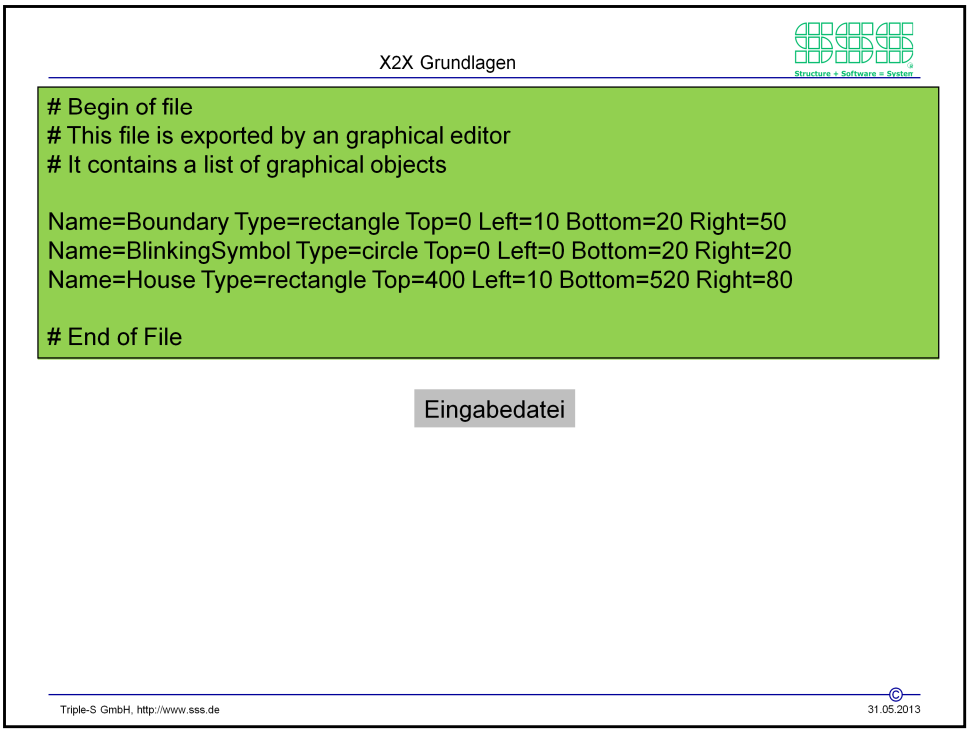
Sind diese Instruktionsdateien einmal erstellt, kann X2X den Vorgang mit gleich strukturierten Eingangsdateien automatisch durchführen und daraus Ausgangsdateien mit entsprechendem Inhalt erzeugen.

Anhand eines einfachen Beispiels werden nachfolgend die Grundlagen von X2X erklärt.

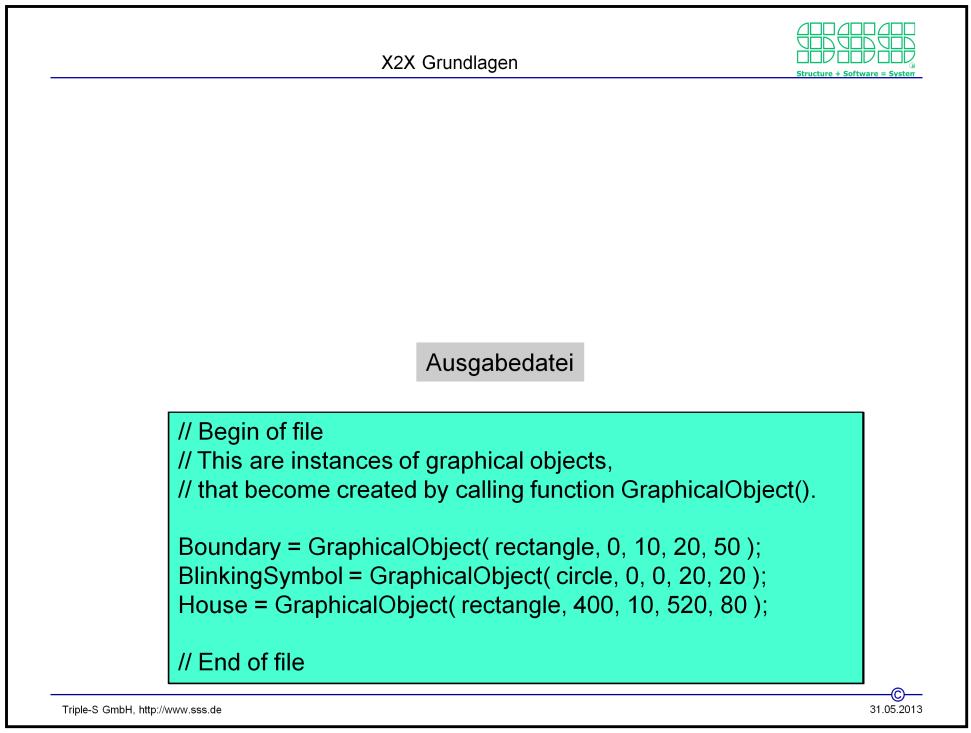


Die einfachste Aufgabenstellung besteht darin, aus genau einer Eingabedatei genau eine Ausgabedatei zu erstellen.

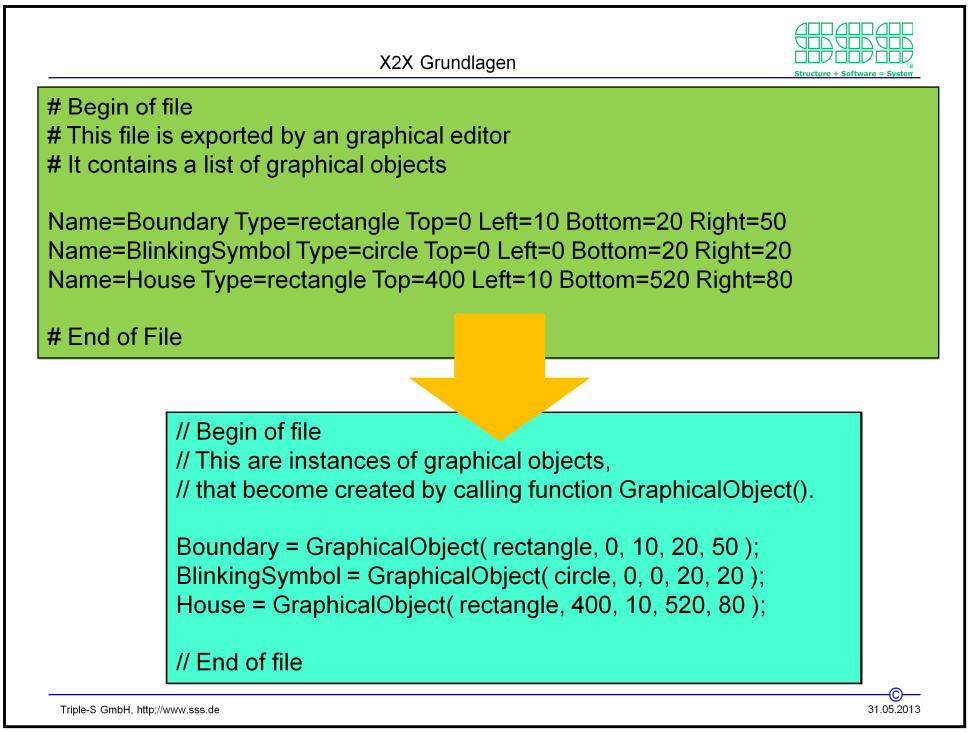
In nachfolgendem Beispiel sind sowohl in der Eingabedatei als auch in der Ausgabedatei identische Nutzdaten vorhanden, allerdings genügen beide Dateien völlig unterschiedlichen Zwecken und Formaten.



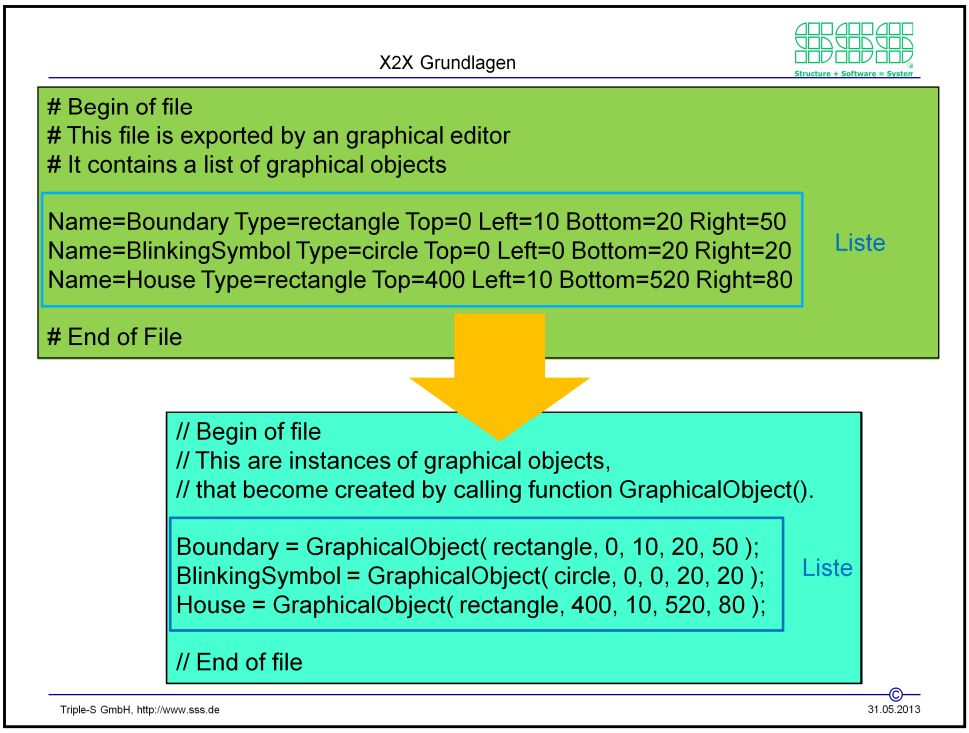
Die Eingabedatei wurde aus einem grafischen Editor exportiert, mit dem drei grafische Objekte (2 Rechtecke und ein Kreis) erfasst wurden.



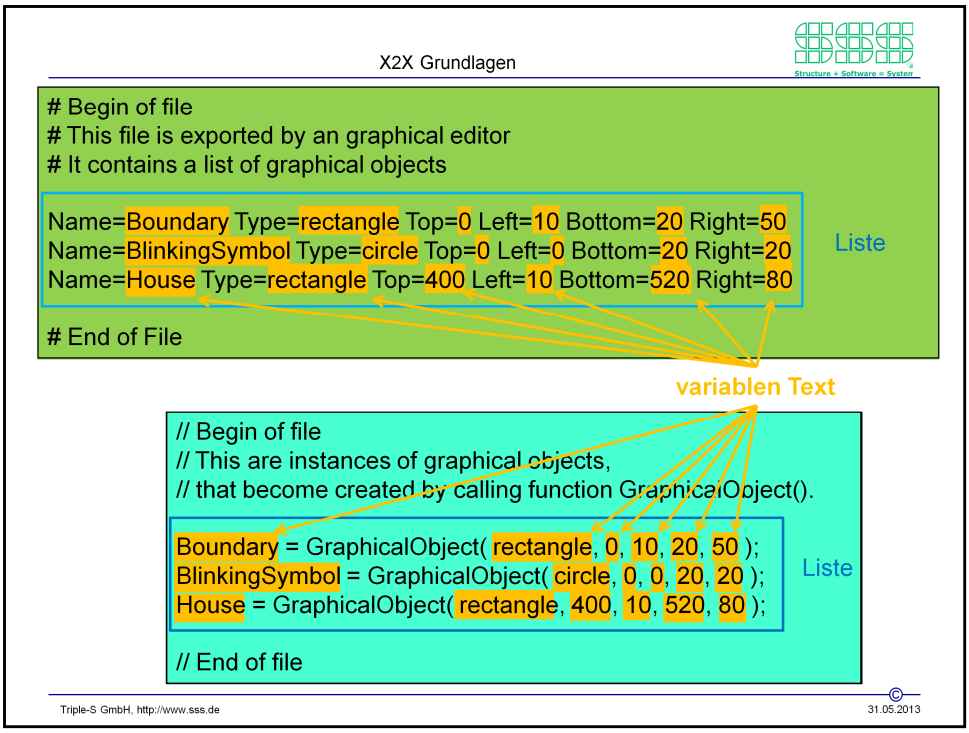
Die Ausgabedatei beschreibt genau die gleichen grafischen Objekte wie die Eingabedatei, allerdings als C Programmiercode.



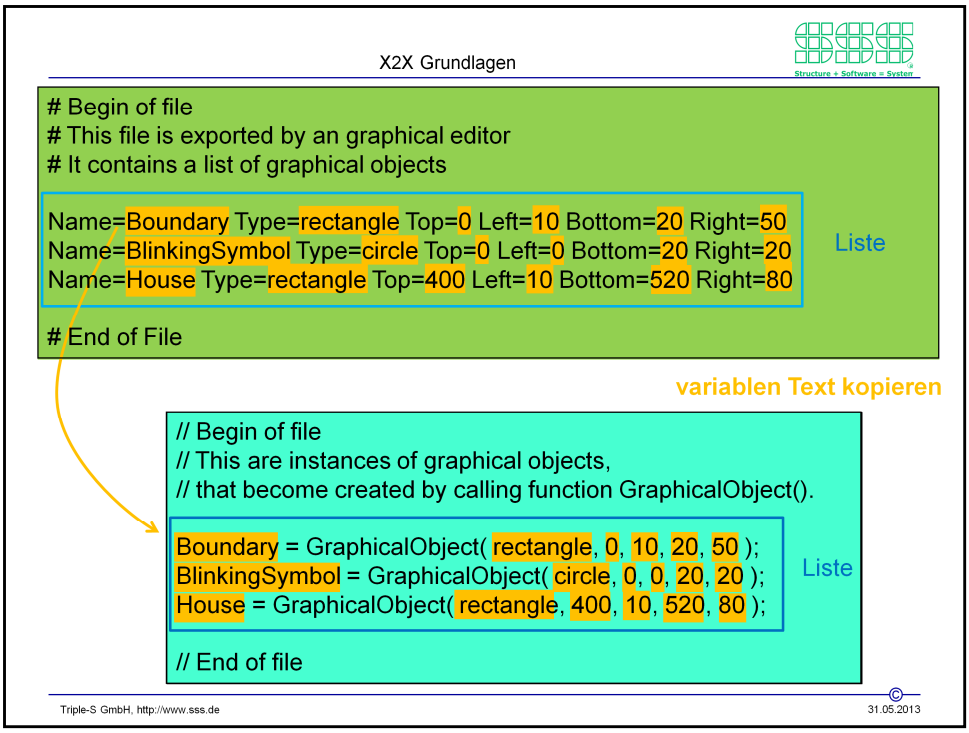
Die in der Praxis vorgegebene Aufgabe könnte also darin bestehen, wann immer eine neue Eingabedatei aus dem grafischen Editor exportiert wurde, den C Programmiercode in der Ausgabedatei entsprechend anzupassen.



Betrachten wir die Ein- und Ausgabedatei genauer, stellen wir fest, dass beide je eine Liste von Daten enthalten, der jeweils ein Block mit festem Text vorausgeht bzw. nachfolgt.



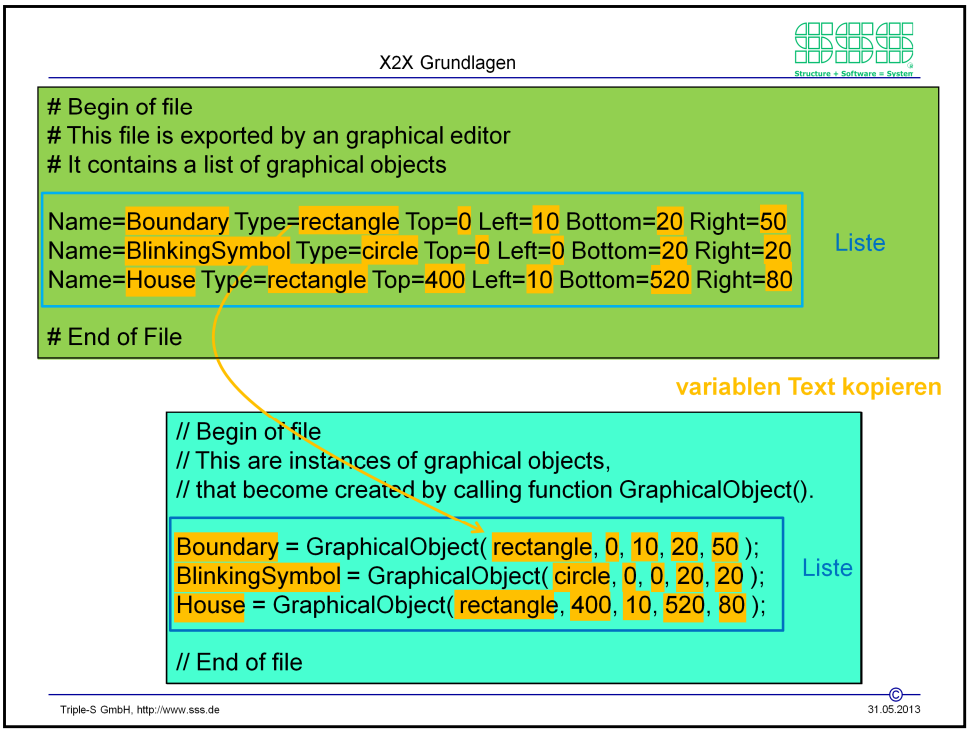
Innerhalb der Liste gibt es feste Texte die unterschiedlich für beide Dateien sind und variable Texte, die für beide Dateien identisch sind.

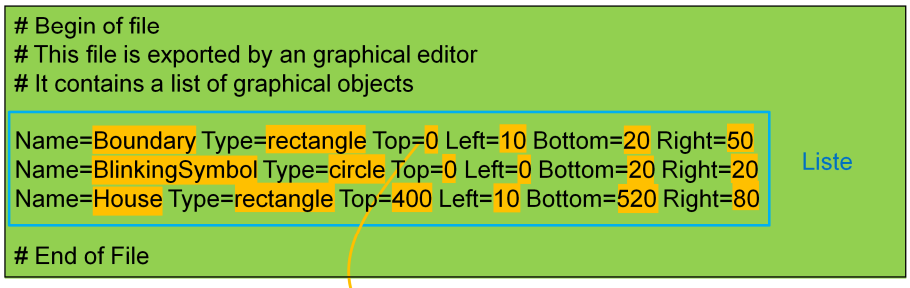


Die variablen Elemente müssen von der Eingabedatei an die entsprechende Stelle in der Ausgabedatei kopiert werden.

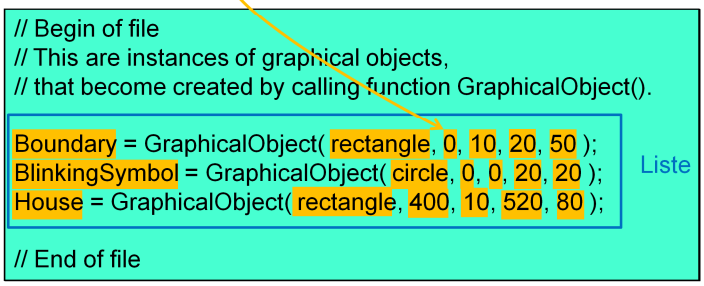
Händisch würde man dies nacheinander mit Copy-und-Paste durchführen.

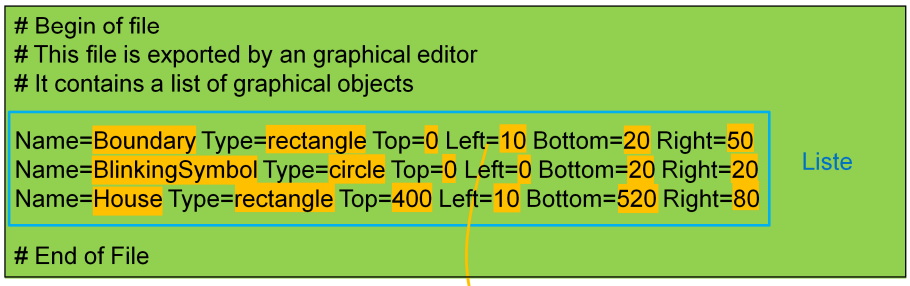
(Es folgt eine Animation mit fünf Bildern für die erste Zeile der Liste.)


# Begin of file
# This file is exported by an graphical editor
# It contains a list of graphical objects
Name=Boundary Type=rectangle Top=0 Left=10 Bottom=20 Right=50
Name=BlinkingSymbol Type=circle Top=0 Left=0 Bottom=20 Right=20
Name=House Type=rectangle Top=400 Left=10 Bottom=520 Right=80
Liste
End of File
variablen Text kopieren
// Begin of file
// This are instances of graphical objects,
// that become created by calling function GraphicalObject().
Boundary = GraphicalObject(rectangle, 0, 10, 20, 50);
BlinkingSymbol = GraphicalObject(circle, 0, 0, 20, 20);
House = GraphicalObject(rectangle, 400, 10, 520, 80);
Liste
// End of file

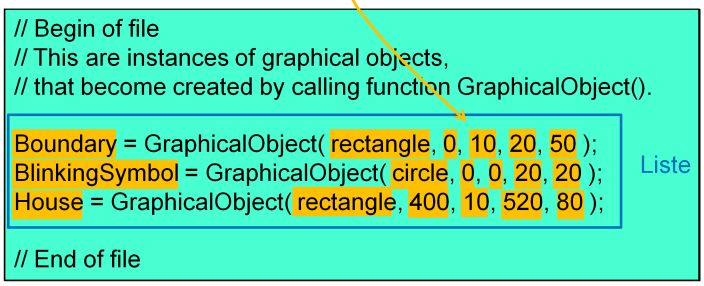


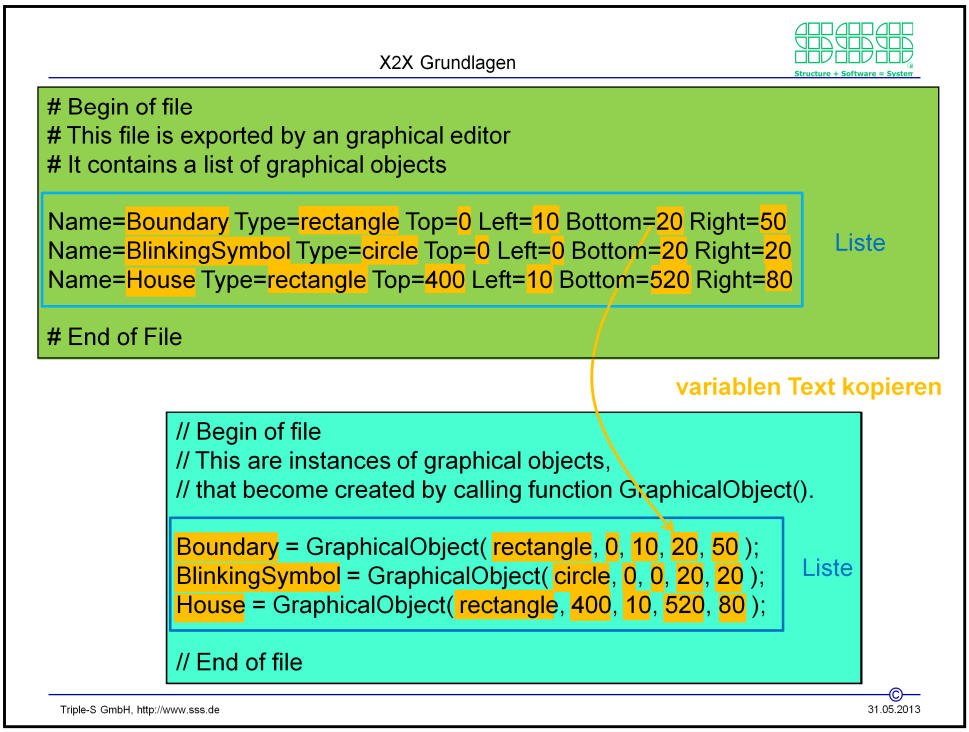
variablen Text kopieren

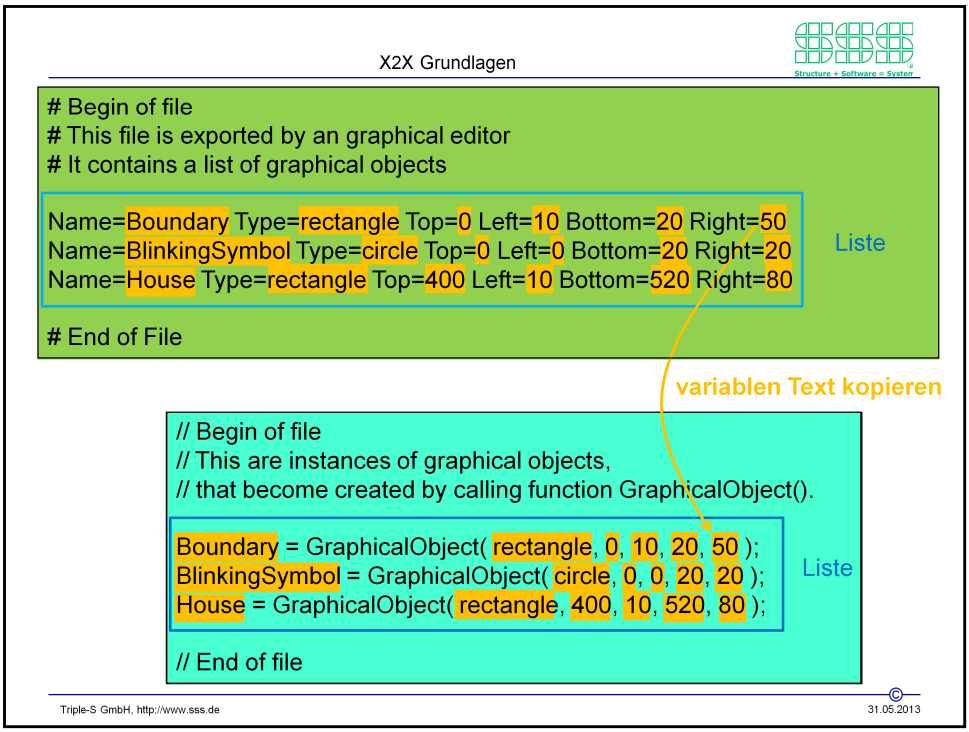


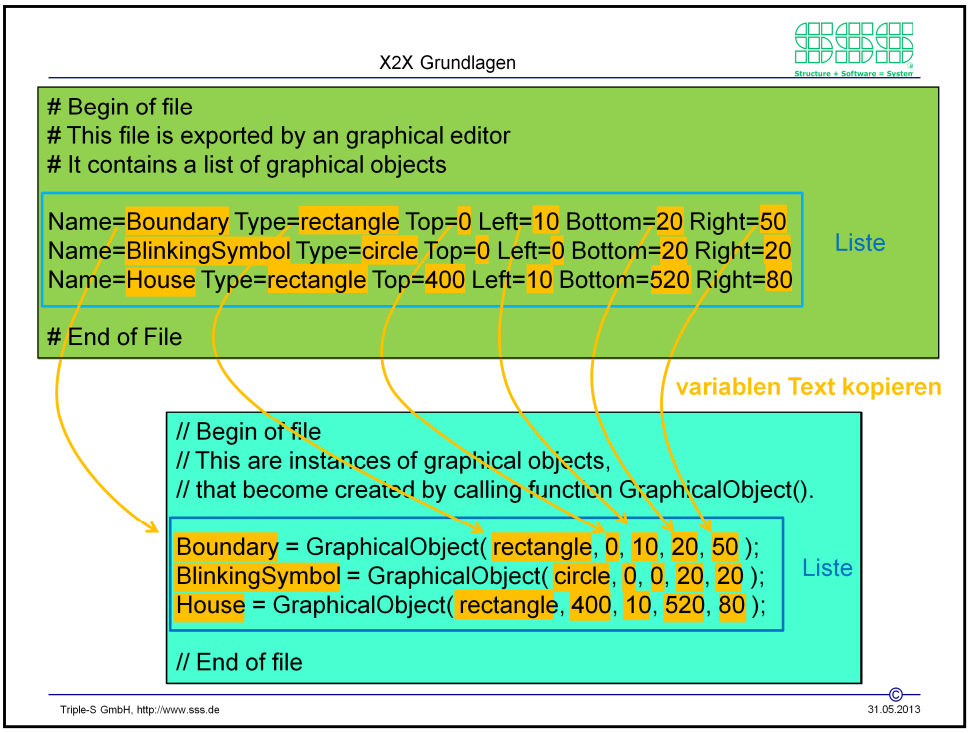


variablen Text kopieren




# Begin of file
# This file is exported by an graphical editor
# It contains a list of graphical objects
Name=Boundary Type=rectangle Top=0 Left=10 Bottom=20 Right=50
Name=BlinkingSymbol Type=circle Top=0 Left=0 Bottom=20 Right=20
Name=House Type=rectangle Top=400 Left=10 Bottom=520 Right=80
Liste
End of File
variablen Text kopieren
// Begin of file
// This are instances of graphical objects,
// that become created by calling function GraphicalObject().
Boundary = GraphicalObject(rectangle, 0, 10, 20, 50);
BlinkingSymbol = GraphicalObject(circle, 0, 0, 20, 20);
House = GraphicalObject(rectangle, 400, 10, 520, 80);
Liste
// End of file

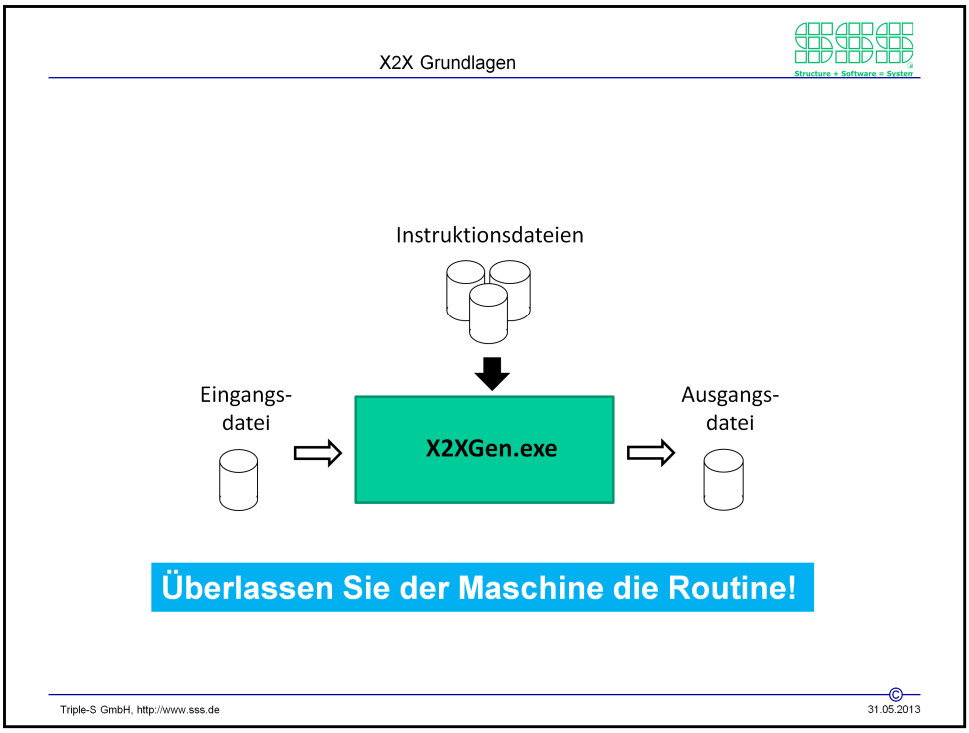

# Begin of file
# This file is exported by an graphical editor
# It contains a list of graphical objects
Name=Boundary Type=rectangle Top=0 Left=10 Bottom=20 Right=50
Name=BlinkingSymbol Type=circle Top=0 Left=0 Bottom=20 Right=20
Name=House Type=rectangle Top=400 Left=10 Bottom=520 Right=80
Liste
End of File
variablen Text kopieren
// Begin of file
// This are instances of graphical objects,
// that become created by calling function GraphicalObject().
Boundary = GraphicalObject(rectangle, 0, 10, 20, 50);
BlinkingSymbol = GraphicalObject(circle, 0, 0, 20, 20);
House = GraphicalObject(rectangle, 400, 10, 520, 80);
Liste
// End of file



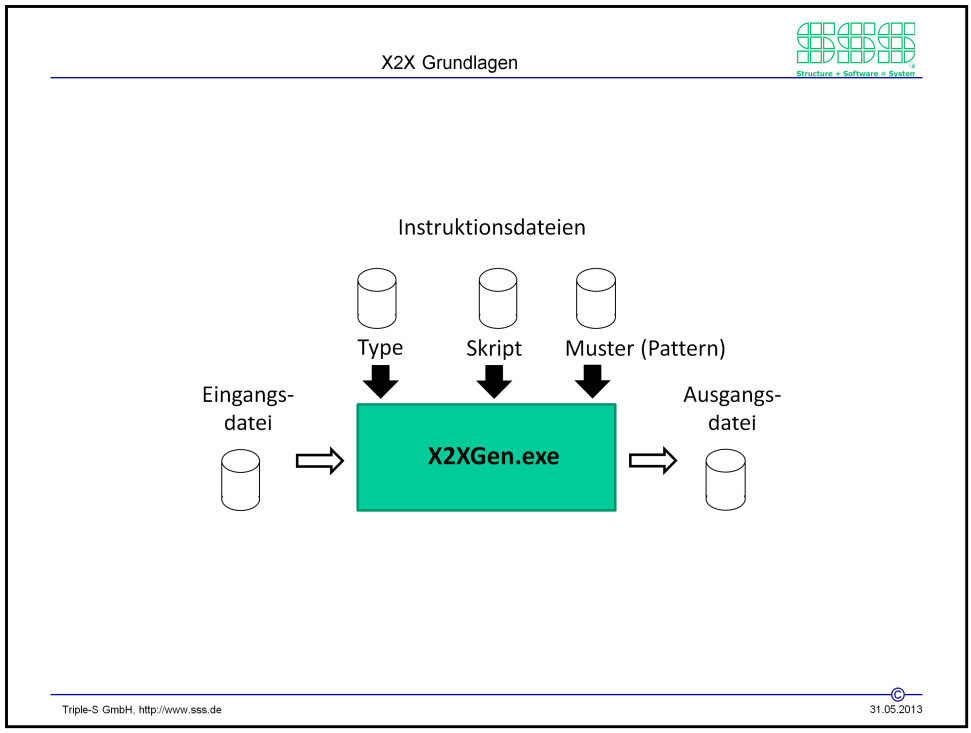
Mit den nächsten beiden Zeilen wird ebenso verfahren.

Natürlich könnte sich auch die Anzahl der Zeilen in der Liste ändern. Dann müssen in der Ausgabedatei entsprechende Zeilen neu erzeugt oder gelöscht werden.

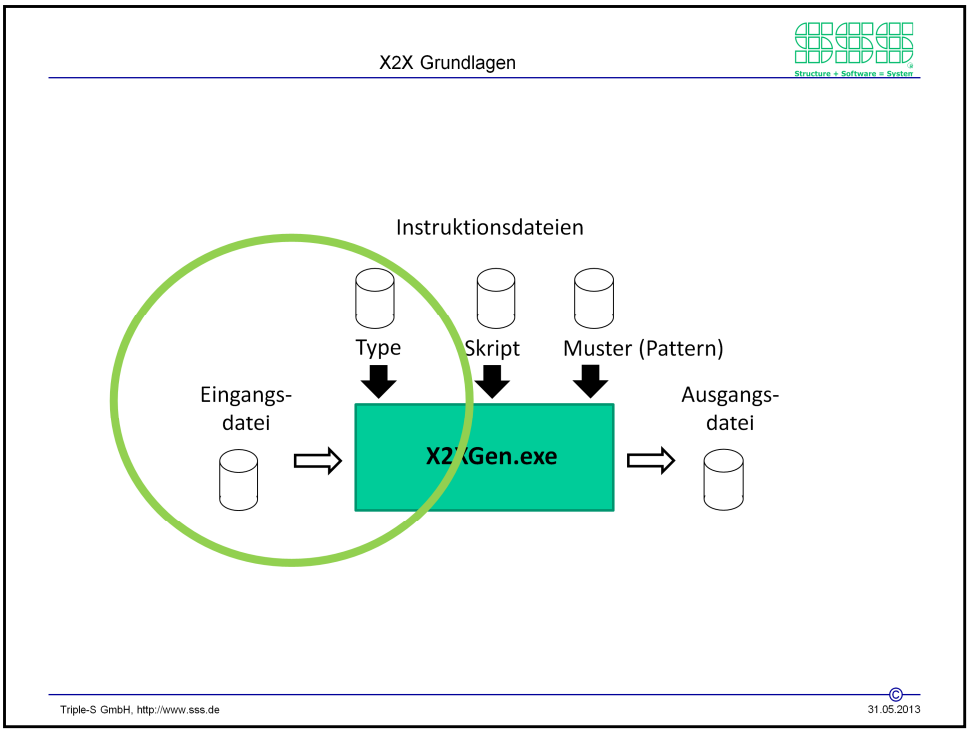
In unserem Beispiel handelt es sich um eine Liste mit nur drei Einträgen. Sowas kann man noch gut von Hand bearbeiten. Aber in der Praxis liegen oft Listen mit mehr als Hundert oder Tausend Einträgen vor.



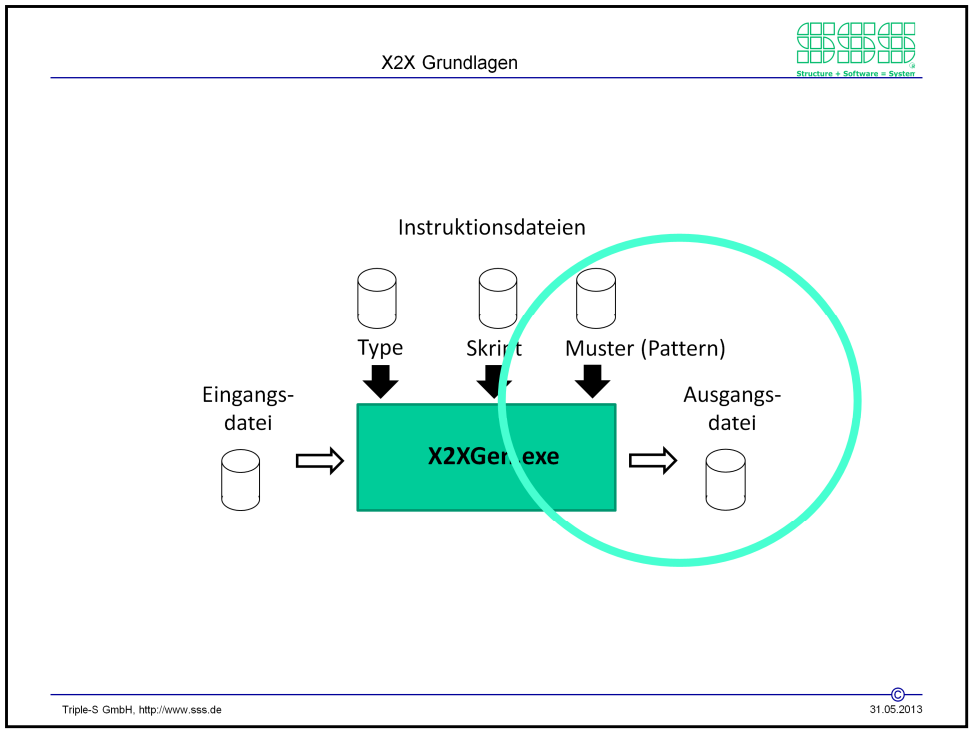
Dann wird es Zeit, diese Routineaufgaben an X2X zu übertragen.



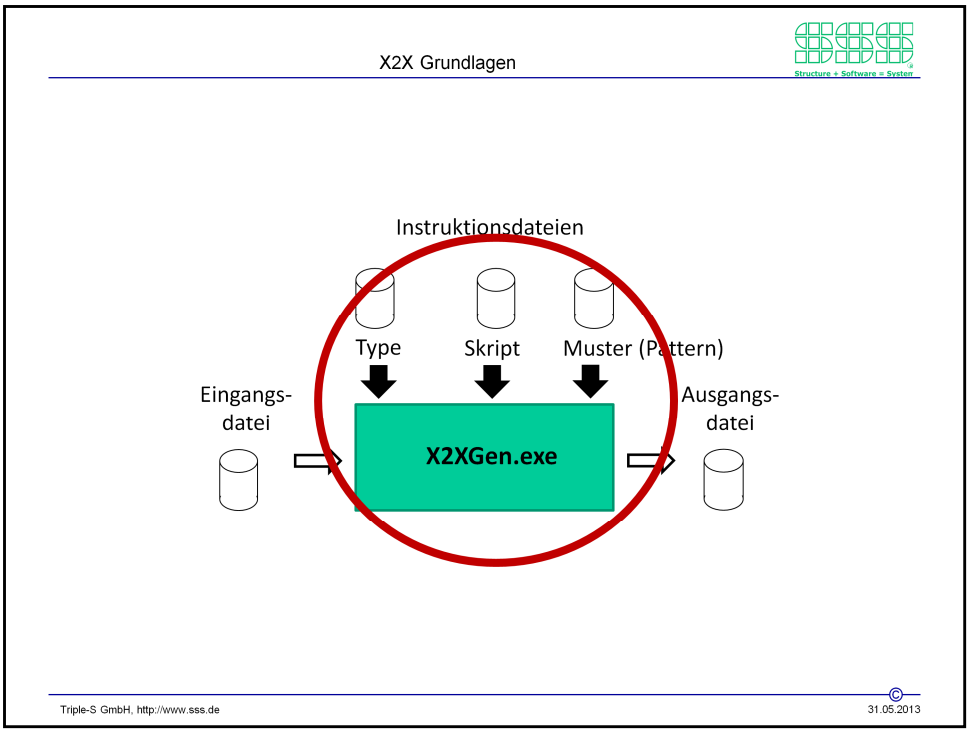
Die Instruktionsdateien mit denen X2X angewiesen wird, was zu tun ist, gliedern sich in drei Arten.



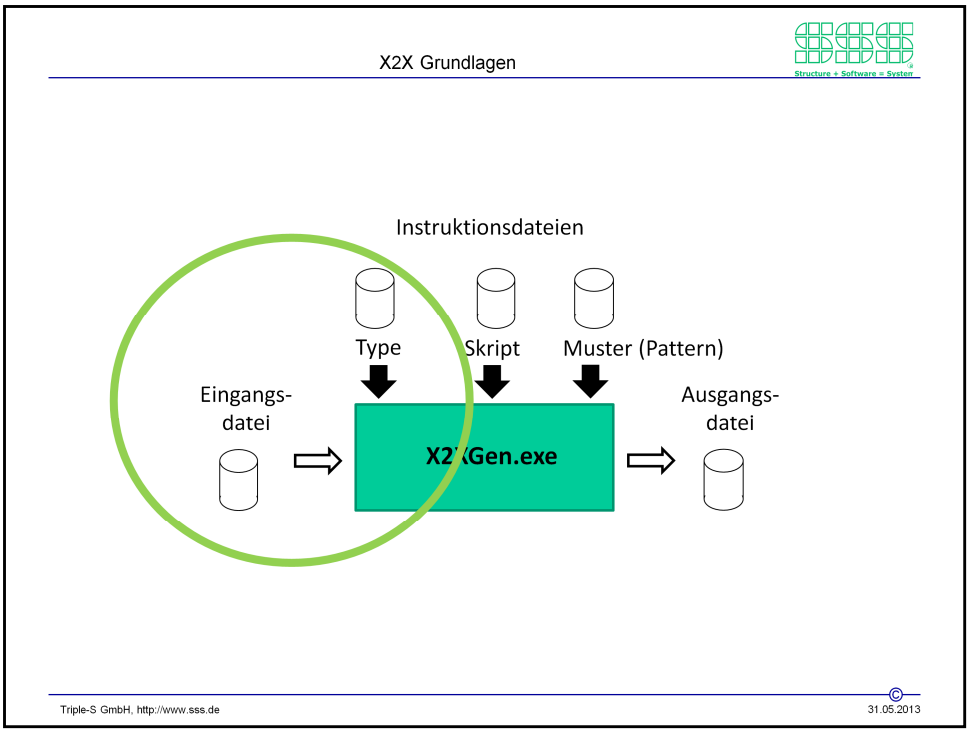
In **Type** Dateien wird festgelegt, wie eine Eingabedatei strukturiert ist, bzw. wo und wie die variablen Elemente gefunden werden können



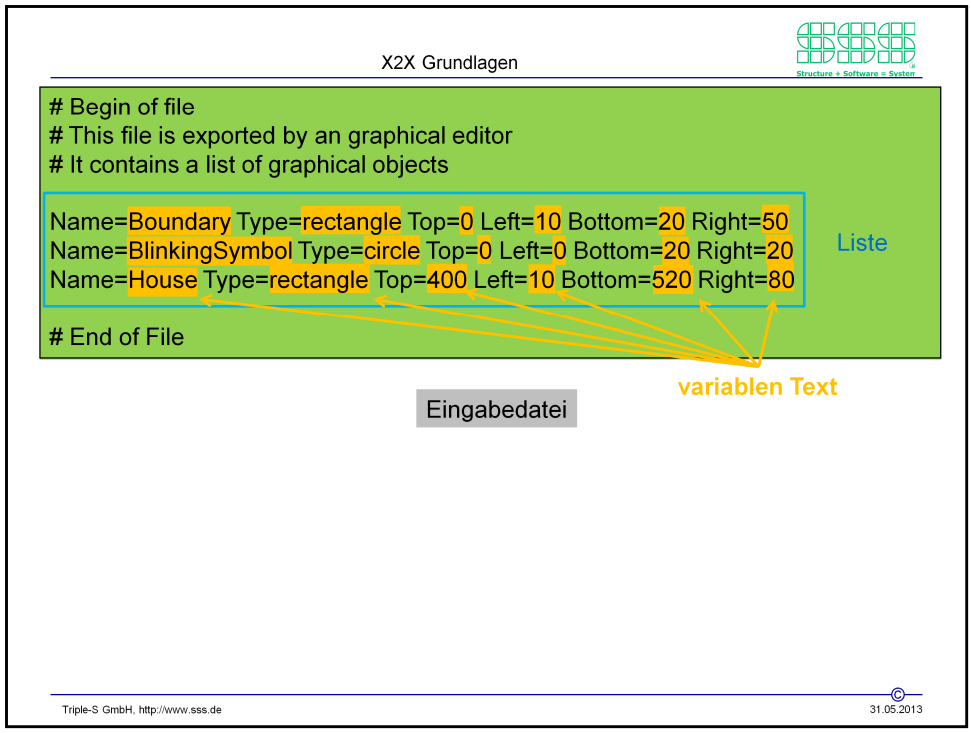
In **Muster** Dateien ist definiert, wie eine Ausgabedatei strukturiert ist bzw. welcher feste Text vorhanden sein muss und wo und wie variable Elemente einzusetzen sind.



In **Skript** Dateien wird festgelegt , welche Eingangsdateien verwendet werden sollen, welcher Type zum einlesen verwendet werden soll, wie die Ausgabedatei heißt und welches Muster zu ihrer Erzeugung verwendet werden soll.

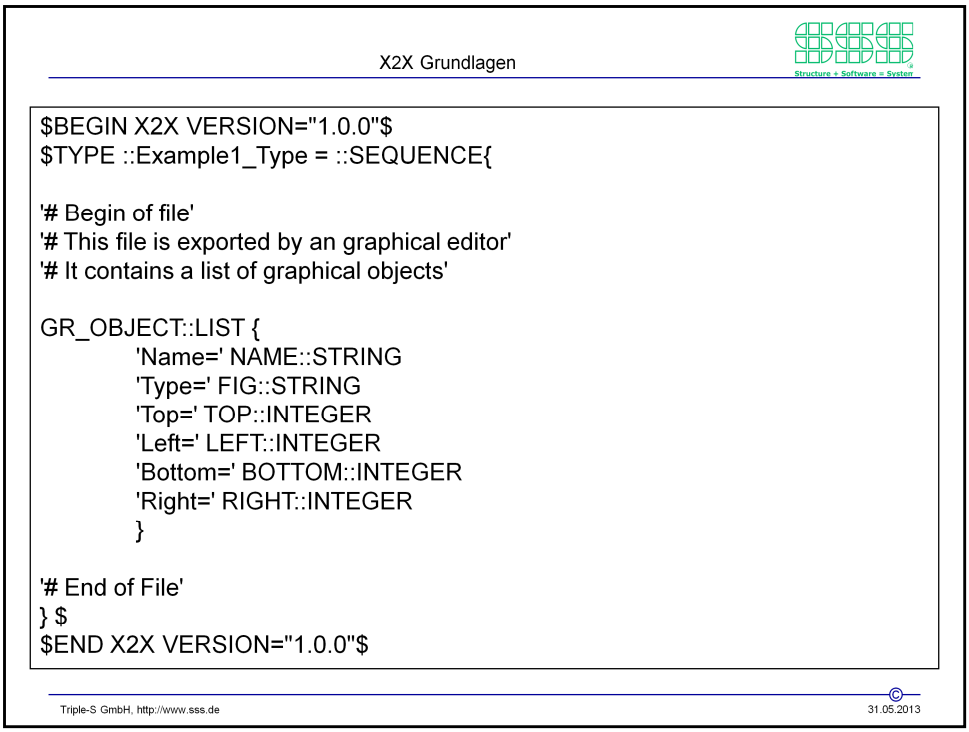


Wir ermitteln zunächst wie die Type-Datei für unser Beispiel aussehen muss.



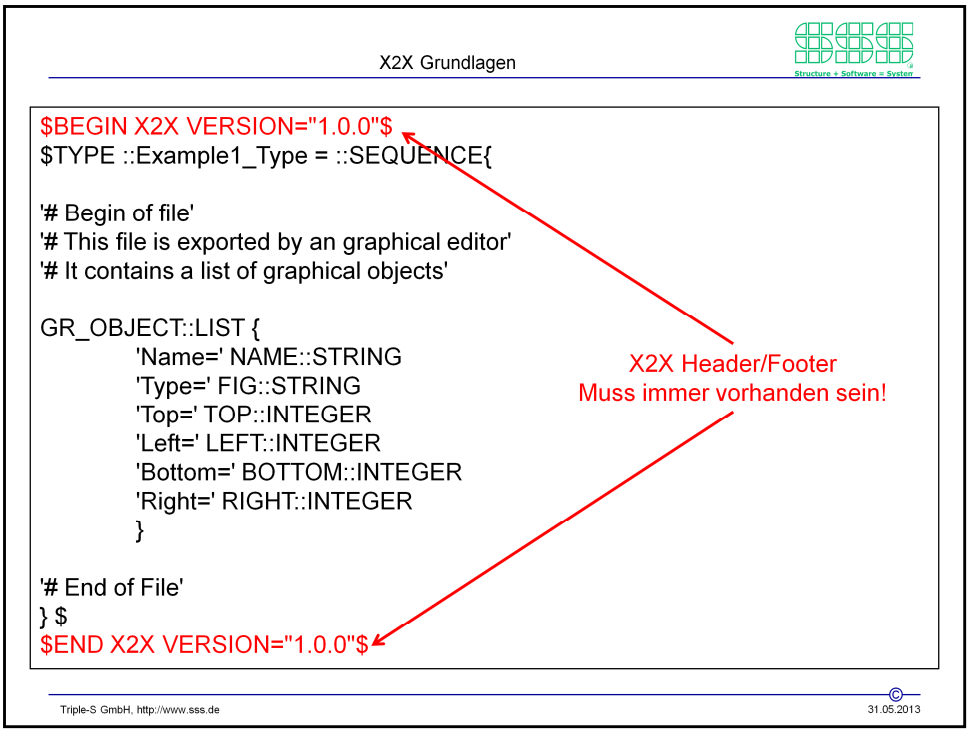
Wir wissen, dass auf den konstanten Einleitungstext die relevante Liste und danach wieder ein konstanter Text folgt.

Die Liste selbst besteht aus konstanten und variablen Elementen.



Dies ist die X2X Type-Datei die wir benötigen.

Nacheinander gehen wir jetzt alle Bestandteile durch.

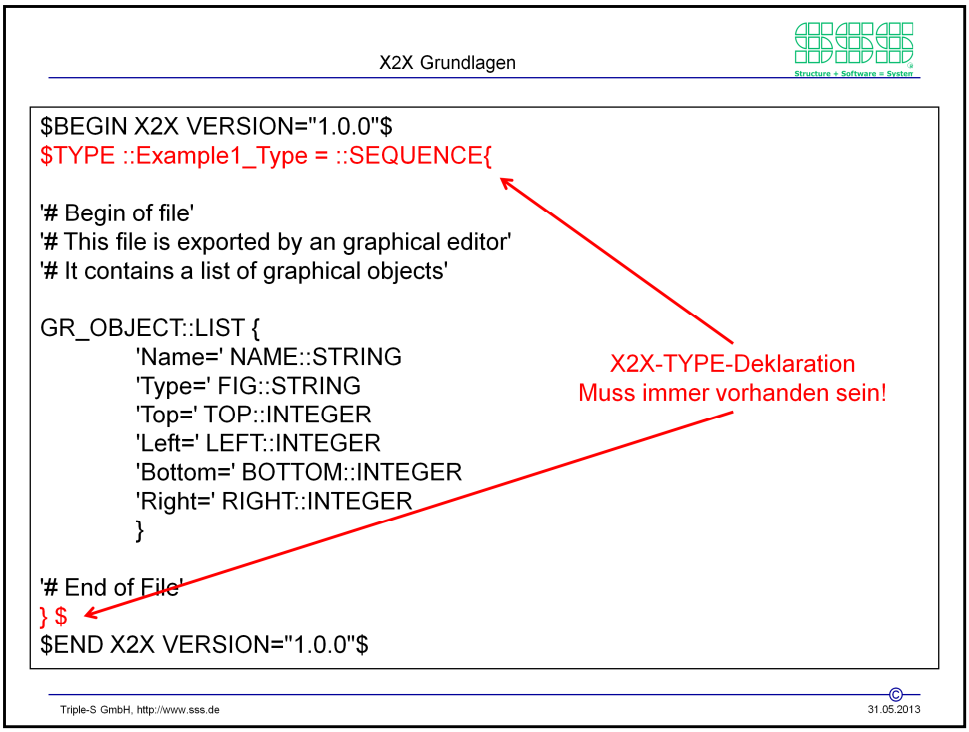


Der X2X Header bzw. Footer muss immer vorhanden sein. Das gilt für alle Arten von X2X Konfigurationsdateien.

Nur zwischen dem X2X Header und X2X Footer können weitere X2X Anweisungen angegeben werden.

X2X Header und X2X Footer sind sogenannte X2X Block-Anweisungen, wobei immer ein Block mit $BEGIN …$ beginnt und mit $END …$ abgeschlossen wird. Ansonsten sind beide Anweisungen immer identisch.

Jede X2X Anweisung beginnt und endet mit einem $ Zeichen!

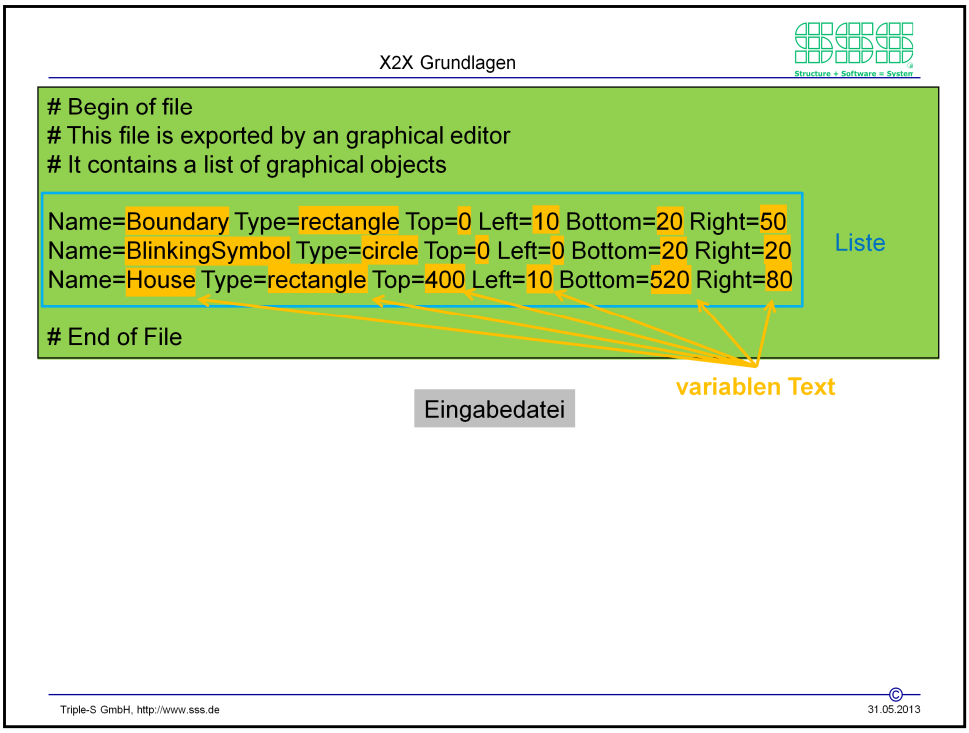


Jede Art von X2X Konfigurationsdatei hat eine für sie typische Anfangs- bzw. Endanweisung.

Bei X2X Type-Dateien ist dies:

$TYPE ::<TypeName> = <TypeDeclaration>$

Innerhalb der <TypeDeclaration> wird der Aufbau des Typen angegeben.

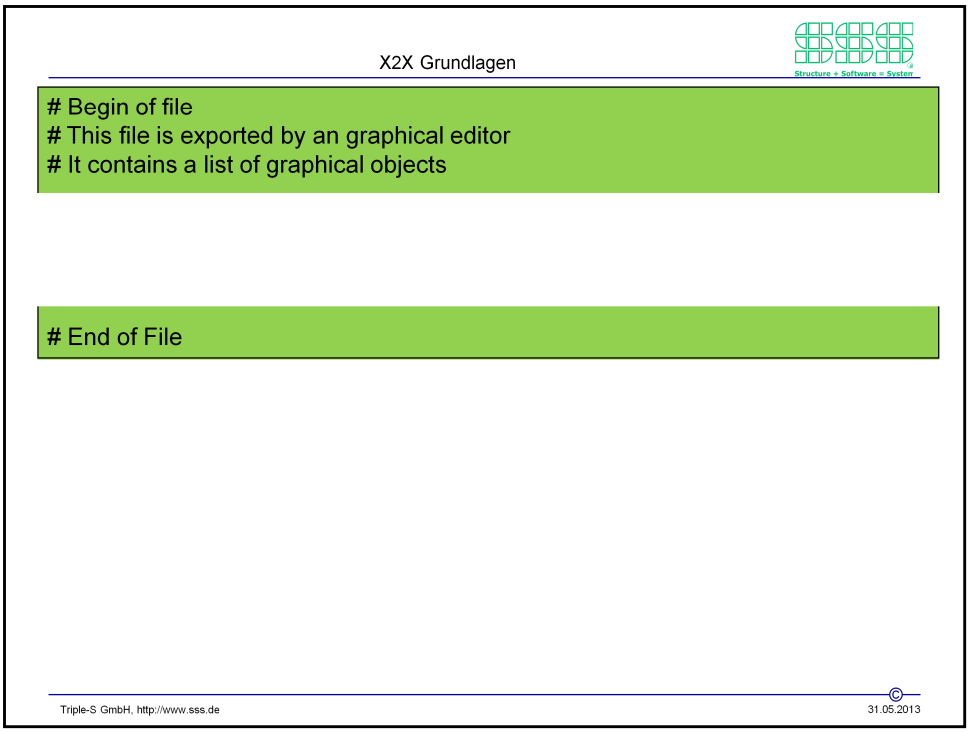


Ein Bick zurück auf die Eingabedatei.

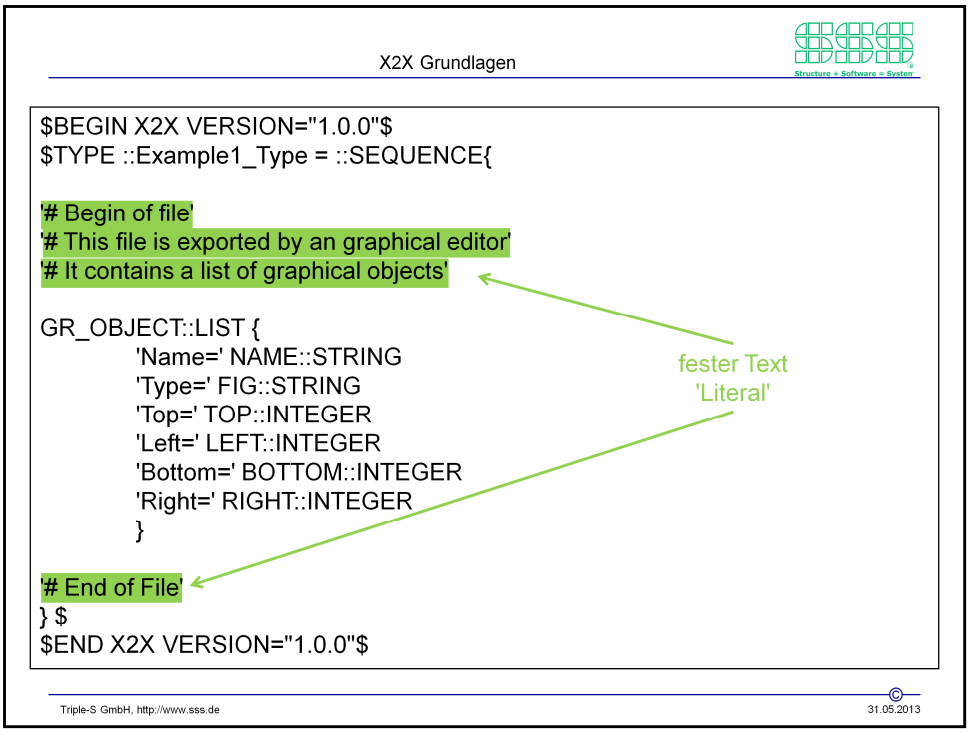
Sie beginnt mit drei konstanten Textzeilen:

'# Begin of file'
'# This file is exported by an graphical editor'
'# It contains a list of graphical objects'

Und endet mit
'# End of File'

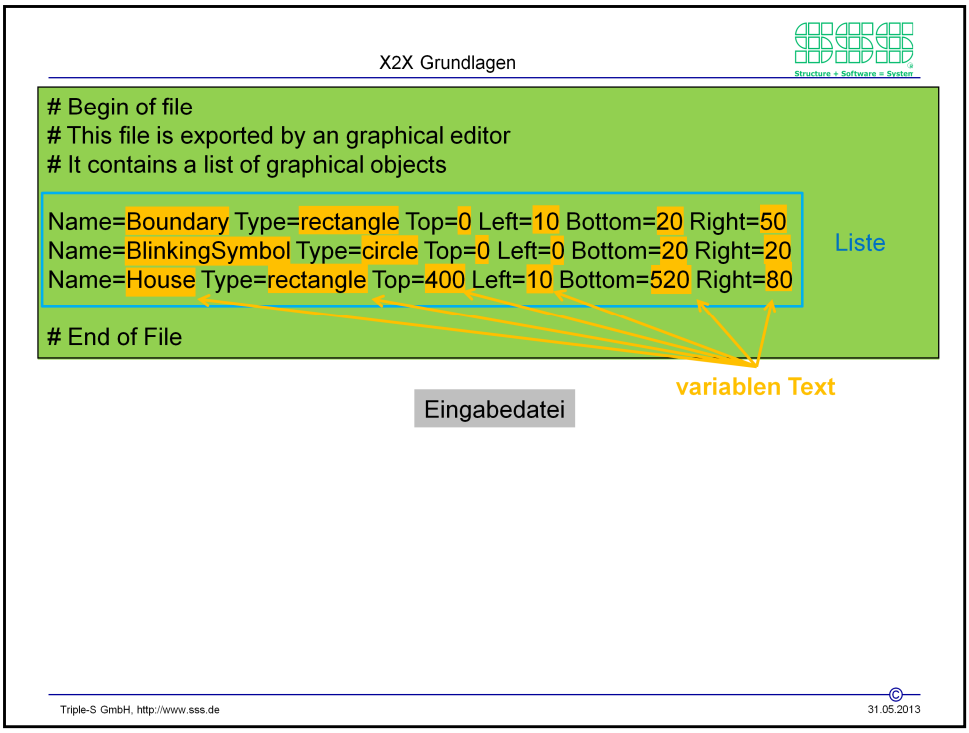


Diesen konstanten Text finden wir in der Typen-Datei wieder.

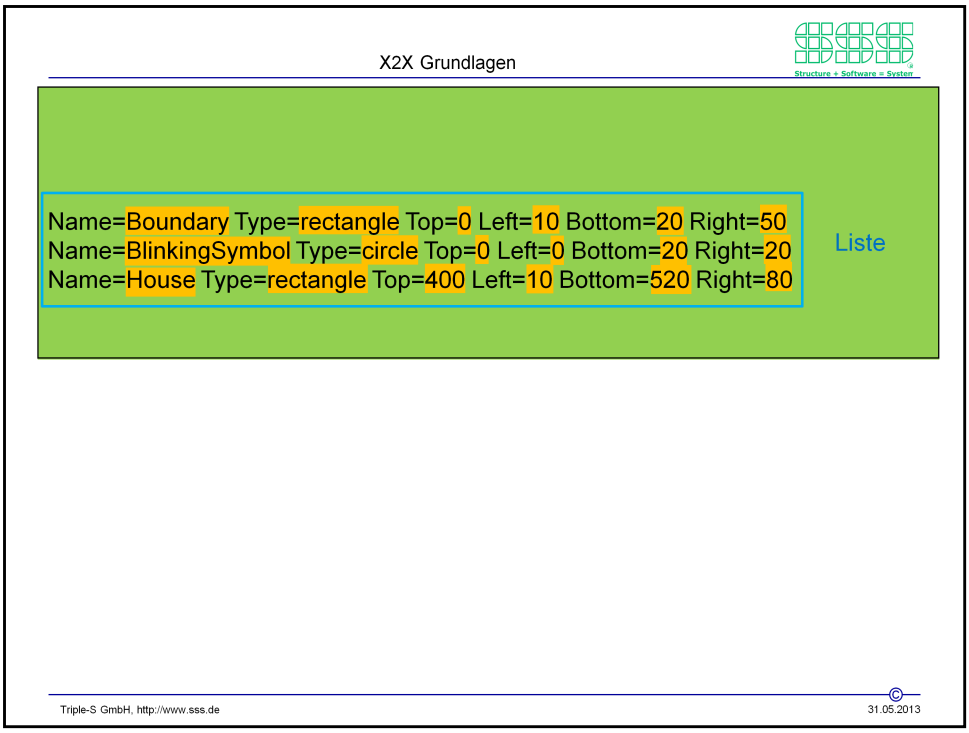


Konstante Texte werden im Typen als Literale mit 'text' angegeben.

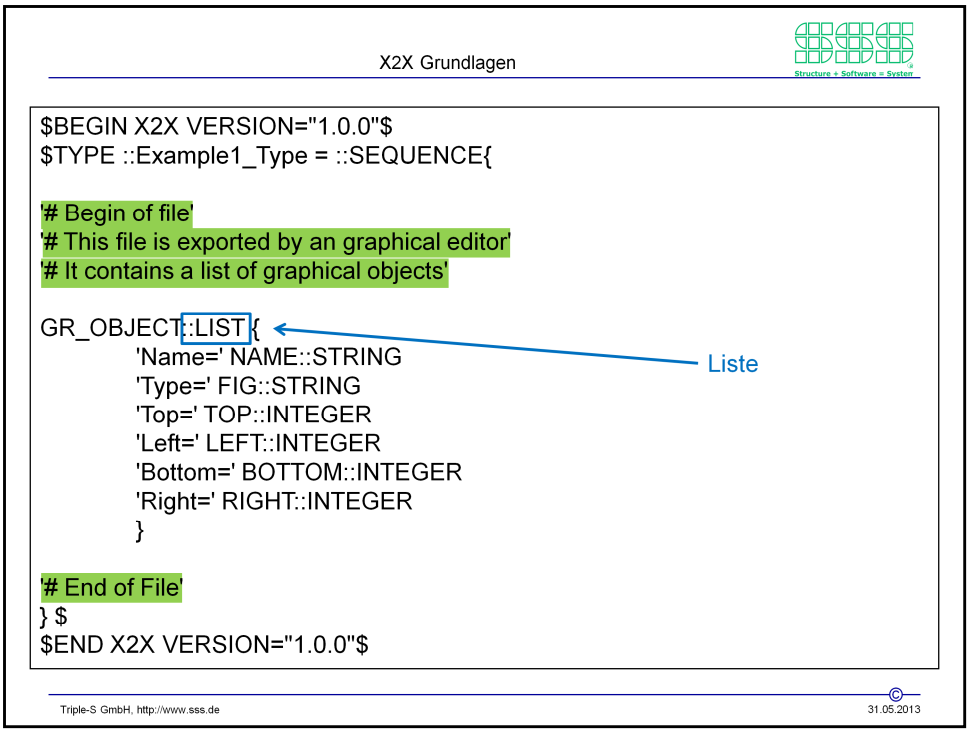
(Man muss nicht notwendigerweise den gesamten konstanten Textblock angeben. Zum Beispiel würde die Deklaration ::STREAM{ 'graphical objects' } X2X mitteilen, dass der nachfolgende konstante Text mit 'graphical objects' endet.)



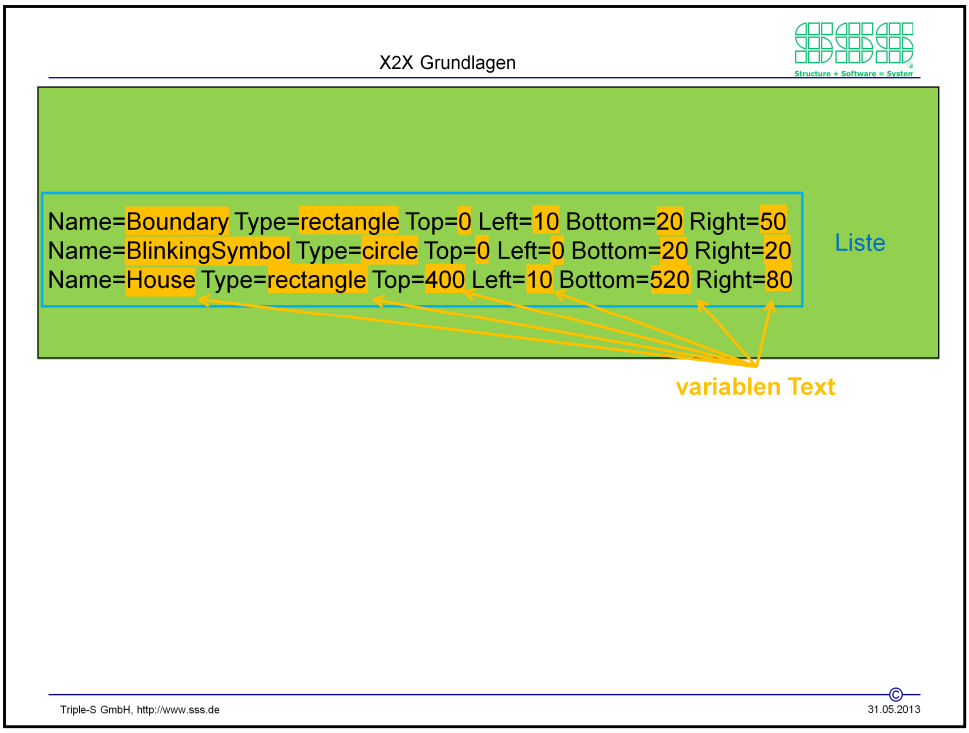
Dem Eingangsblock mit konstantem Text folgt eine Liste.



Diese Liste müssen wir im Type angeben …

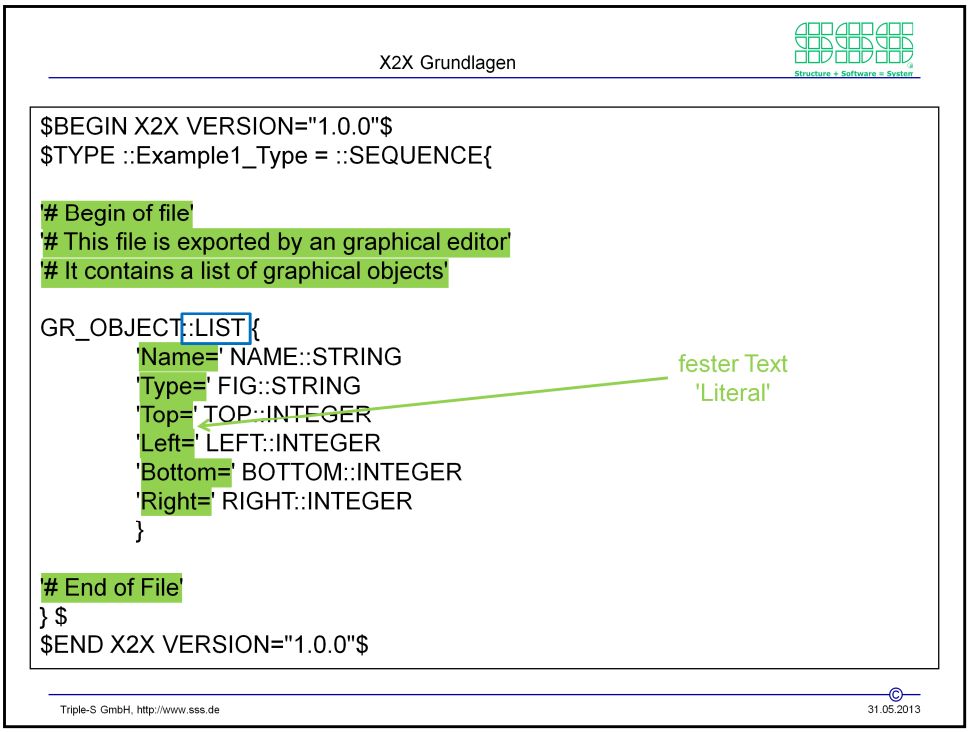


::LIST ist eine X2X Strukturkategorie, die besagt, dass alle zwischen den anschließenden Klammern { und } stehenden Elemente sich als Gruppe beliebig oft wiederholen können, (aber mindestens einmal vorhanden sind.)

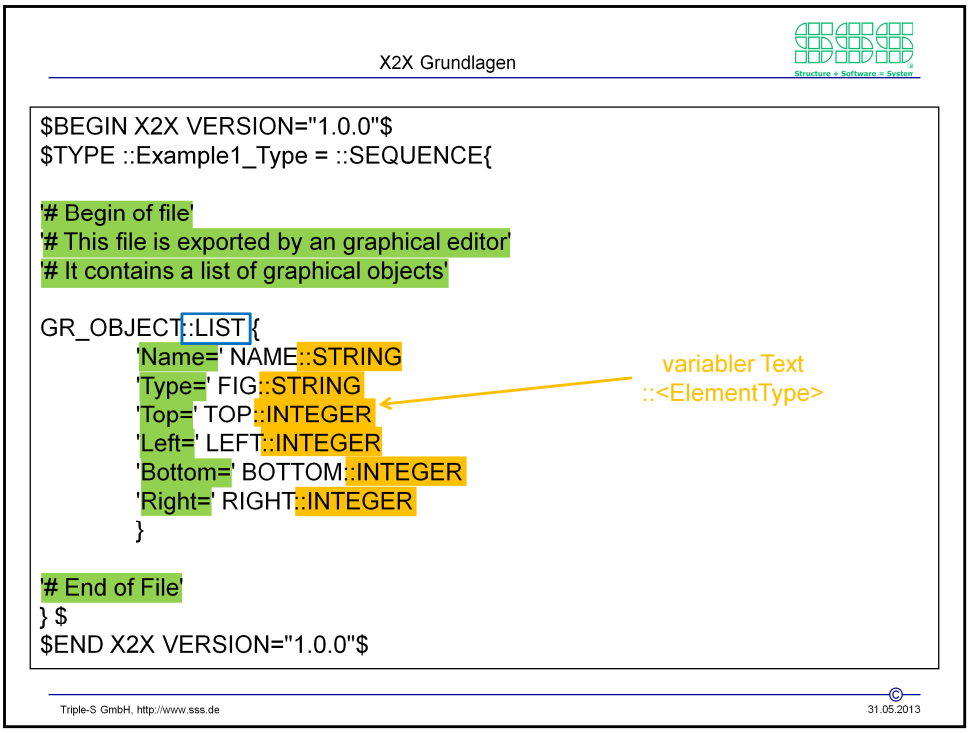


Die Liste enthält konstante und variable Elemente.

Die variablen Elemente sind entweder Strings (Text) oder Integer (ganze Zahlen).



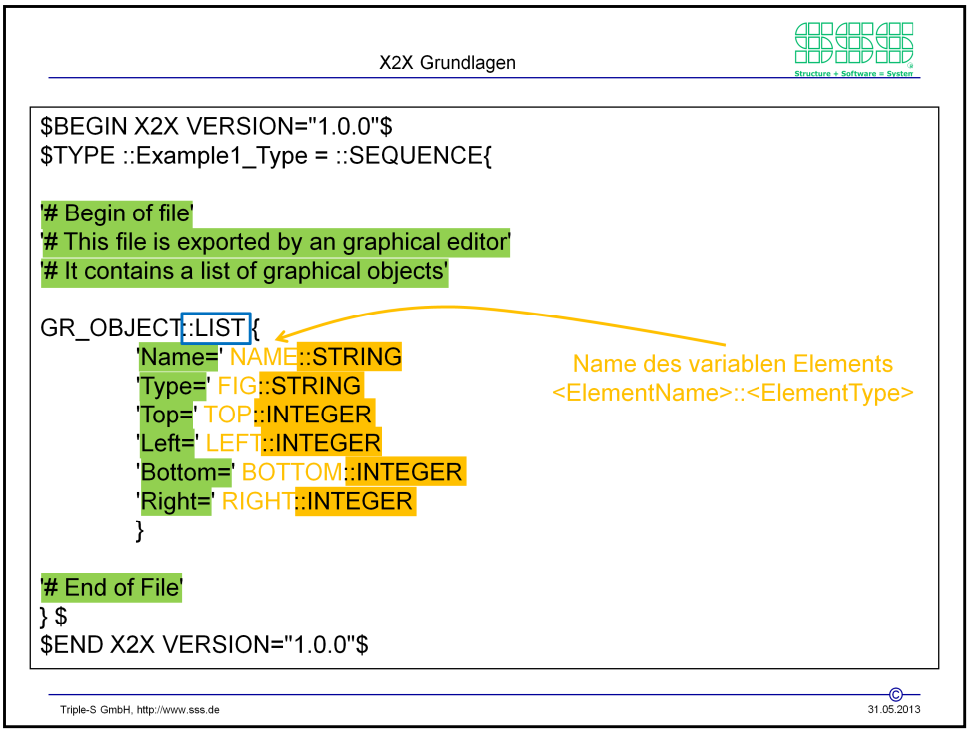
Die konstanten Elemente werden wieder als Literale angegeben.



Die variablen Elemente werden mit einem Datentype ::<ElementType> angegeben.

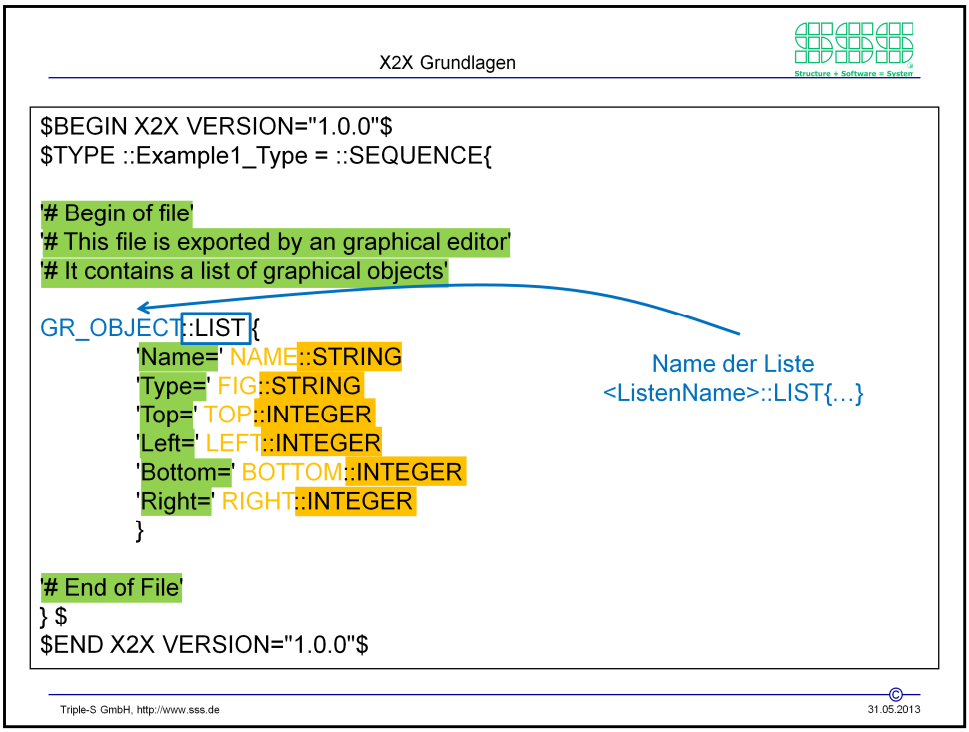
::STRING ist eine elementarer Type, der aus einer beliebigen Kombination von sichtbaren Zeichen besteht.

::INTEGER ist eine positive oder negative ganze Zahl oder 0.



Bei variablen Elementen werden in der Regel dem Datentype mit <ElementName>::<ElementType> ein Name für das Element vorangestellt. Über diesen Namen kann in einem Skript oder Muster auf den Wert des variablen Elements zugegriffen werden.

Will man später auf ein variables Element nicht zugreifen, benötigt es keinen Namen.

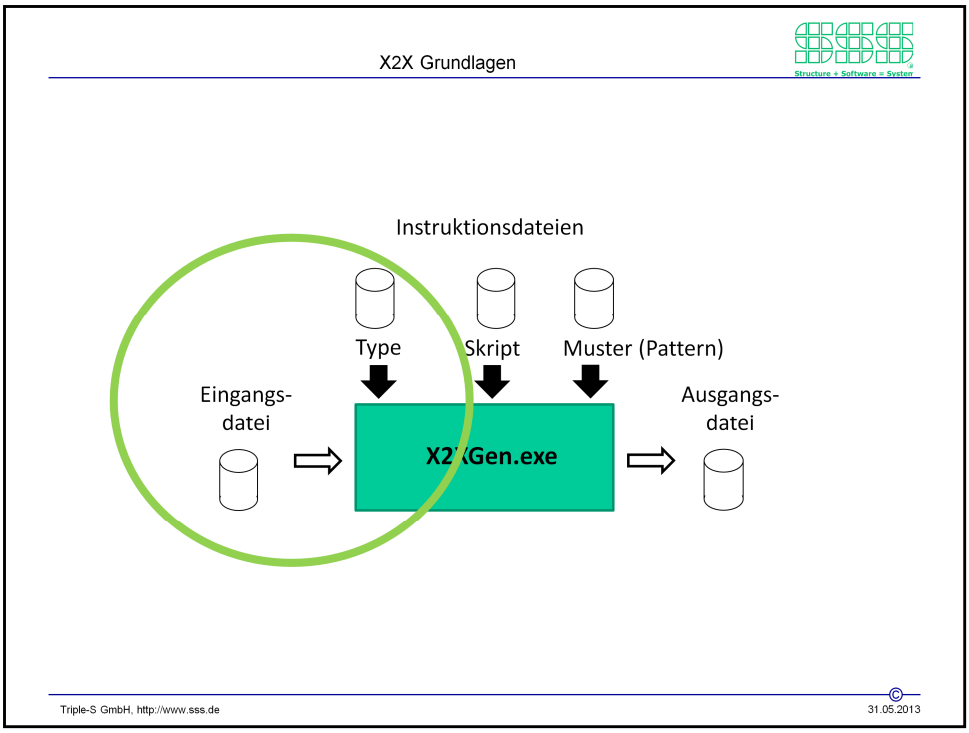


Analog ist GR_OBJECT der Name der Liste.

```
$BEGIN X2X VERSION="1.0.0"$
$TYPE ::Example1_Type = ::SEQUENCE{

'# Begin of file'
'# This file is exported by an graphical editor'
'# It contains a list of graphical objects'

GR_OBJECT::LIST{
            'Name=' NAME::STRING
            'Type=' FIG::STRING
            'Top=' TOP::INTEGER
            'Left=' LEFT::INTEGER
            'Bottom=' BOTTOM::INTEGER
            'Right=' RIGHT::INTEGER
            }

'# End of File'
} $
$END X2X VERSION="1.0.0"$
```

Fertig ist der Type!

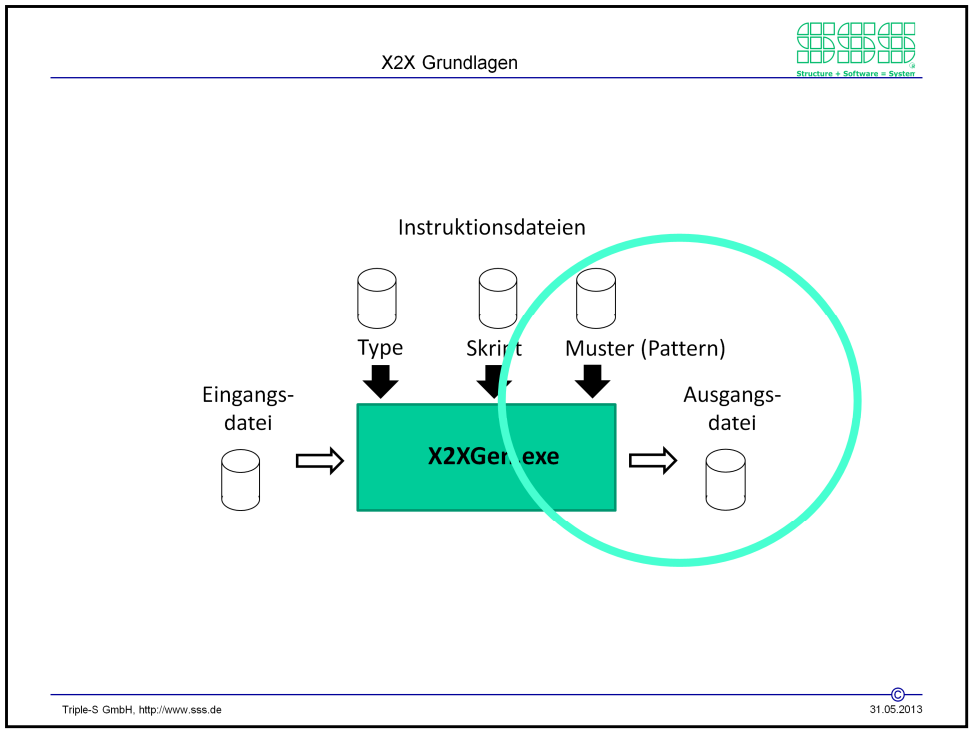
Fertig ist der Type, der die Struktur unserer Eingabedatei beschreibt.



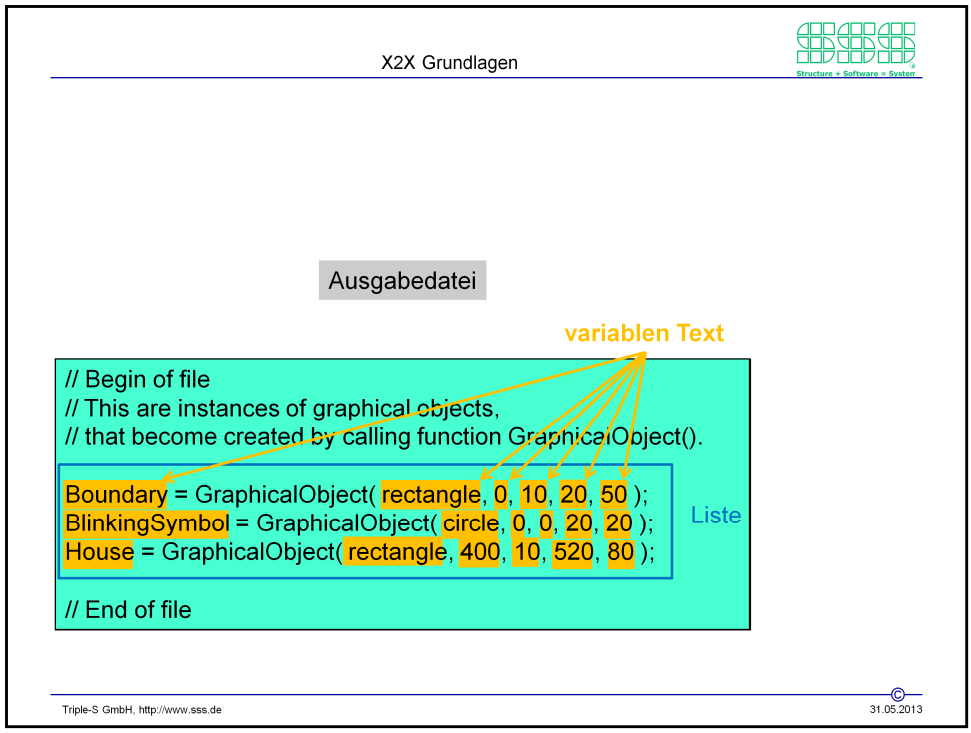
Mit dem Type haben wir die Struktur unserer Eingabedatei beschrieben.

<u>Alle variablen Elemente haben einen Namen, mit dem auf ihren Wert zugegriffen werden kann!</u>

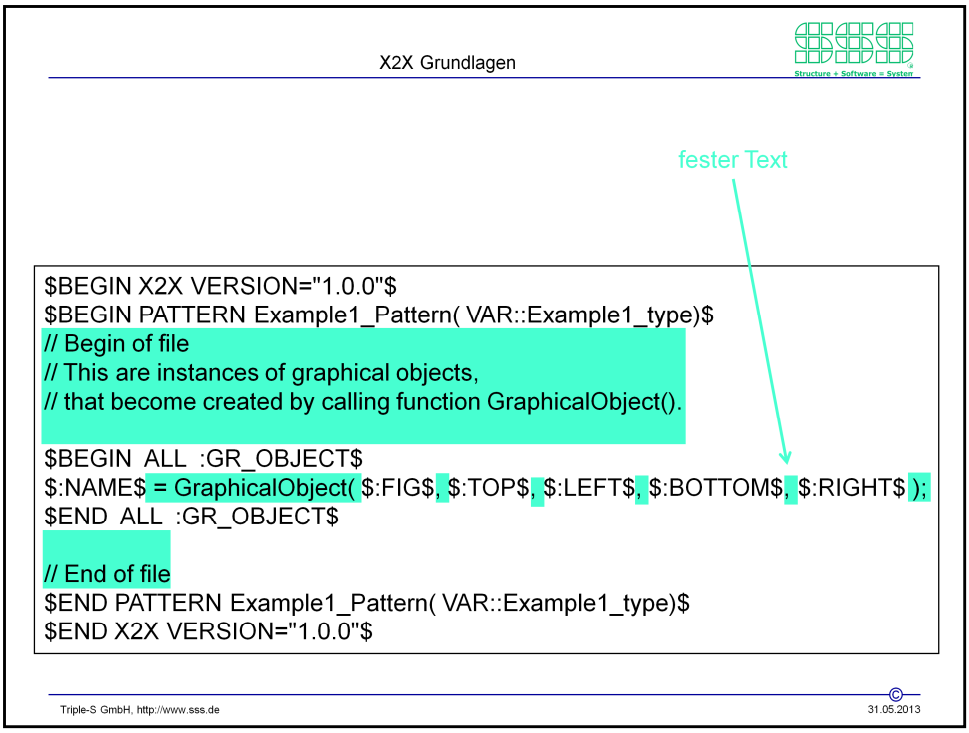
Diese verwenden wir im Muster zur Ausgabe der Werte der variablen Elemente.



Jetzt erstellen wir für unsere Ausgabedatei ein Muster (Pattern).



Die Datei, die wir erzeugen wollen, besteht überwiegend aus konstantem Text und einer Liste mit variablen Elementen.

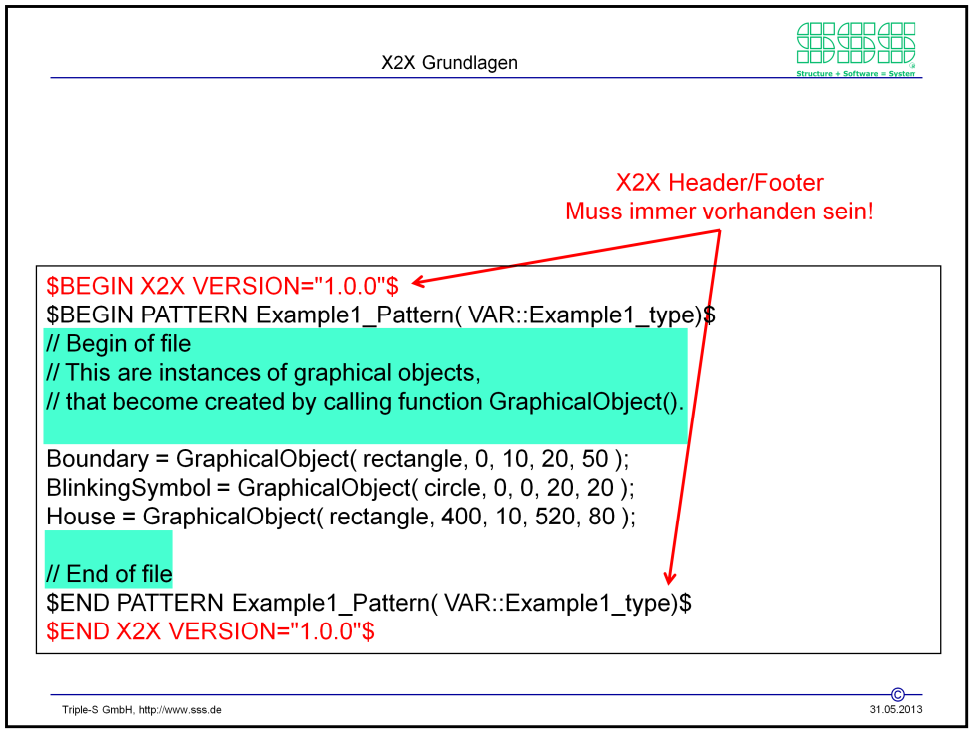


Dies ist die fertige Muster-Datei die wir benötigen.

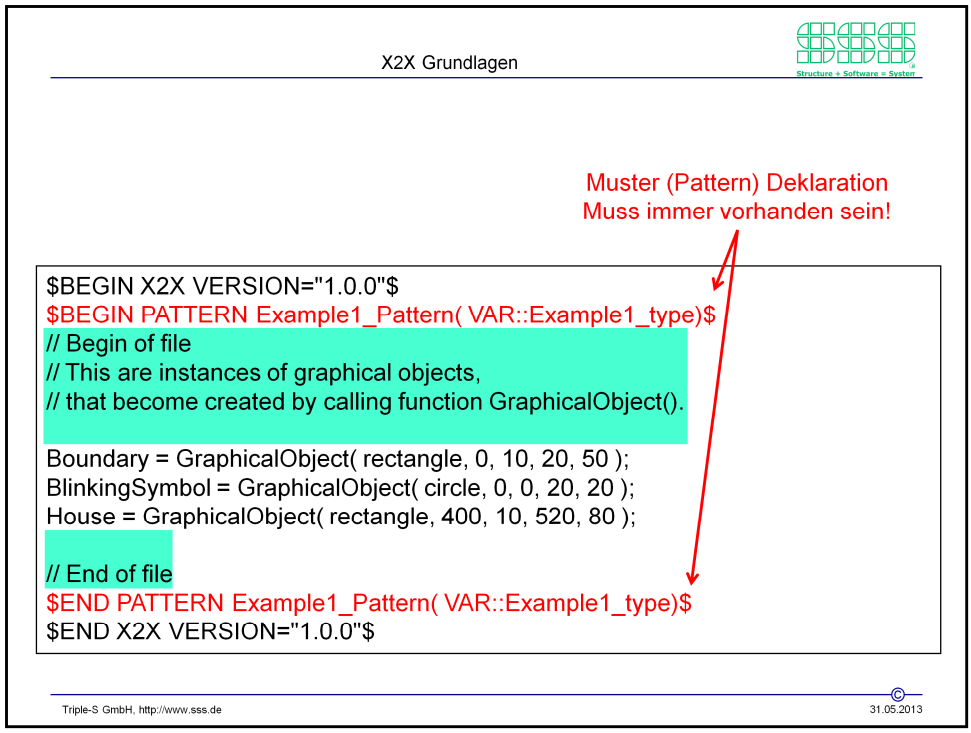
Für das Erstellen einer Muster-Datei gilt das WYSIWYG-Prinzip. D.h., der konstante Text wird mit allen Zwischenräumen und Zeilenumbrüchen genau so angegeben, wie er später in der Ausgabedatei stehen soll.

WYSIWYG heißt ausgeschrieben: „What You See Is What You Get“
Zu deutsch: „Was Du siehst, ist was Du bekommst“

In der Praxis erstellt man eine Muster-Datei tatsächlich so, dass man von einer Kopie der Ausgabedatei startet und den X2X Header/Footer hinzufügt.



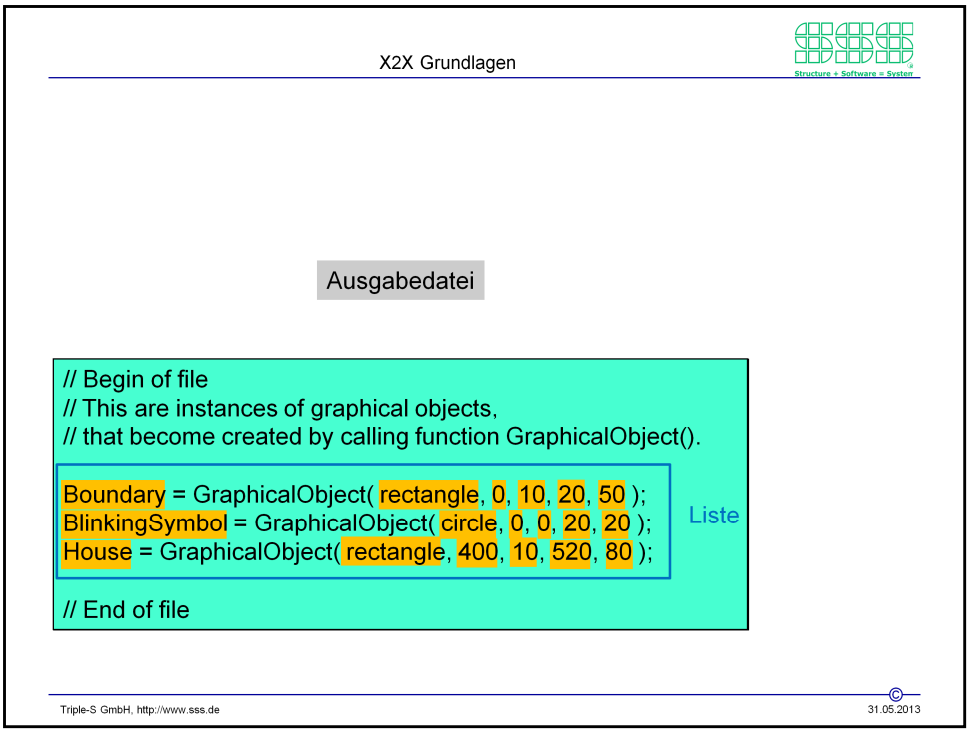
Dies ist eine Kopie der Ausgabedatei, die um den notwendigen X2X Header und Footer erweitert wurde.



Ebenso hinzugefügt wurde die notwendige Muster (Pattern) Deklaration, die X2X drüber informiert, dass es sich um eine Muster-Datei handelt.

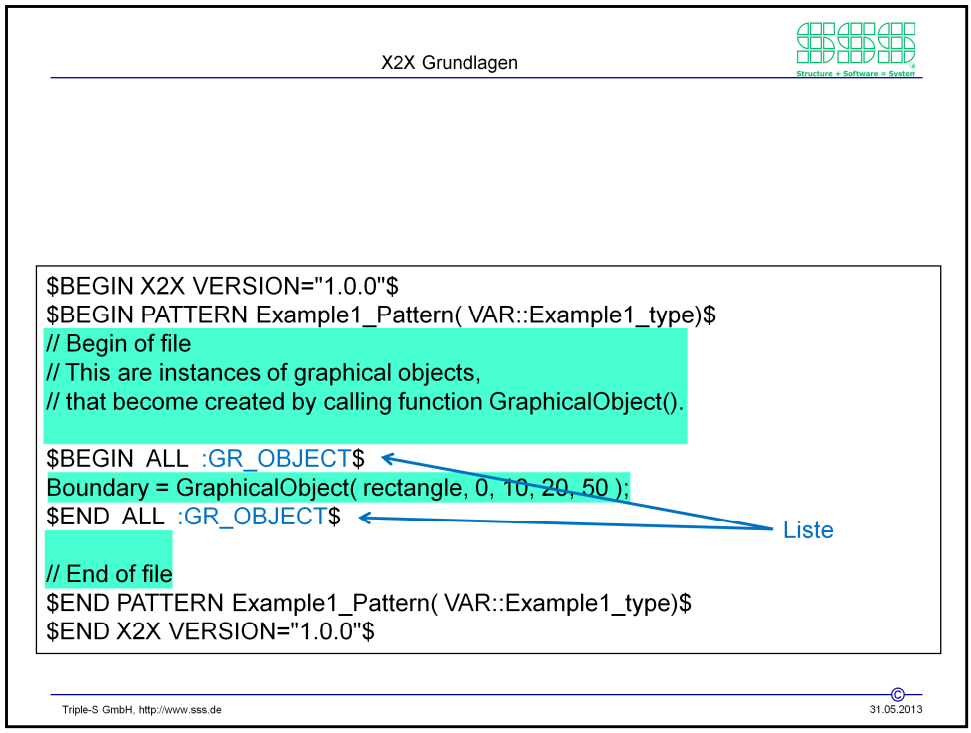
Durch Hinzufügen dieser vier Zeilen haben wir eine gültige Muster-Datei erstellt, die genau die Ausgabedatei erzeugt von der wir ausgegangen sind.

Das ist aber nicht der Sinn der Sache, da wir ja die Ausgabedatei in Abhängigkeit von den variablen Elementen der Eingabedatei erstellen wollen!



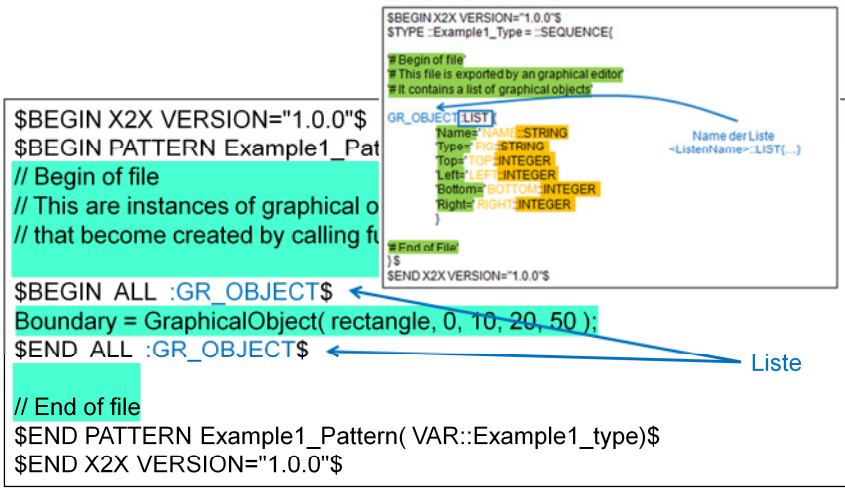
Zur Erinnerung:

Die Ausgabedatei enthält eine Liste.

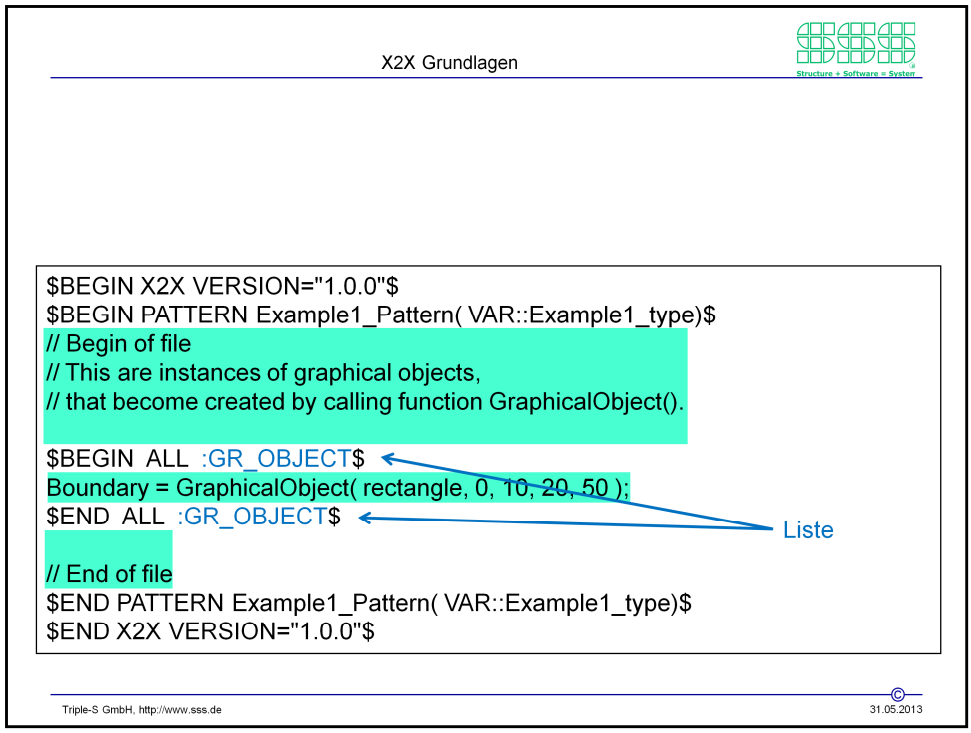


Für jede Zeile in der Liste der Eingabedatei soll in der Ausgabedatei ebenfalls eine Zeile erzeugt werden.

Dies wird durch die X2X Blockanweisung $BEGIN ALL :GR_OBJECT$ erreicht.



GR_OBJECT war der Name den wir in der Type-Datei für die Liste festgelegt haben.

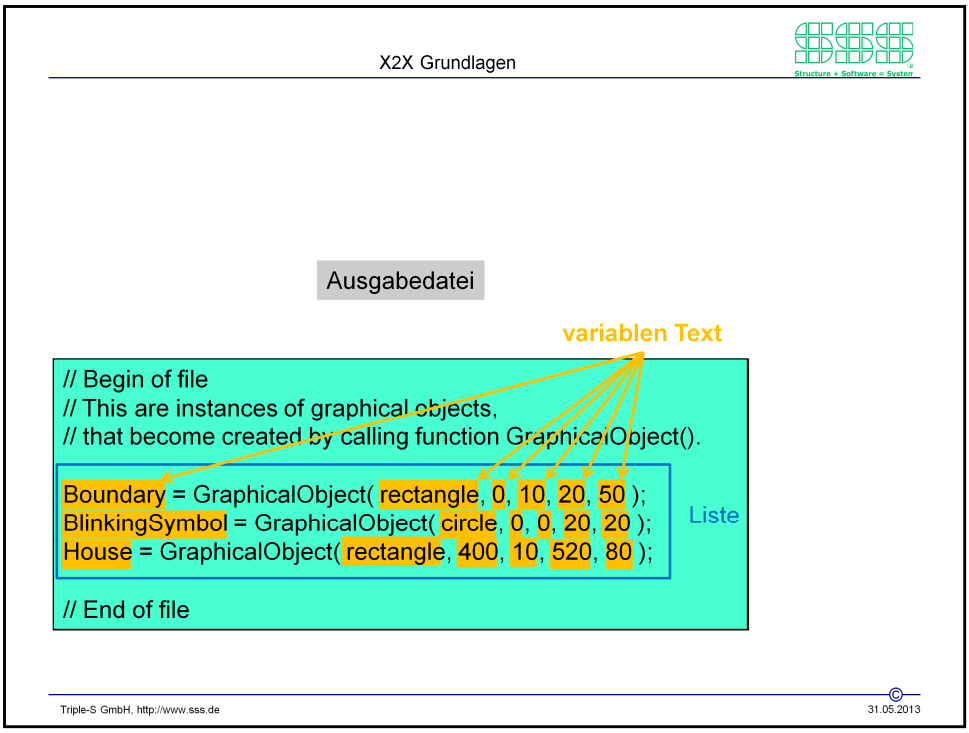


Jeder konstante Text und jede X2X Anweisung zwischen den X2X Blockanweisungen $BEGIN ALL :GR_OBJECT$ und $END ALL :GR_OBJECT$ wird für alle Zeilen der Liste :GR_OBJECT einmal ausgeführt.

D.h. mit dieser Muster-Datei würden wir in der Ausgabedatei drei Zeilen mit identischem konstantem Text erhalten.

Boundary = GraphicalObject(rectangle, 0, 10, 20, 50);
Boundary = GraphicalObject(rectangle, 0, 10, 20, 50);
Boundary = GraphicalObject(rectangle, 0, 10, 20, 50);

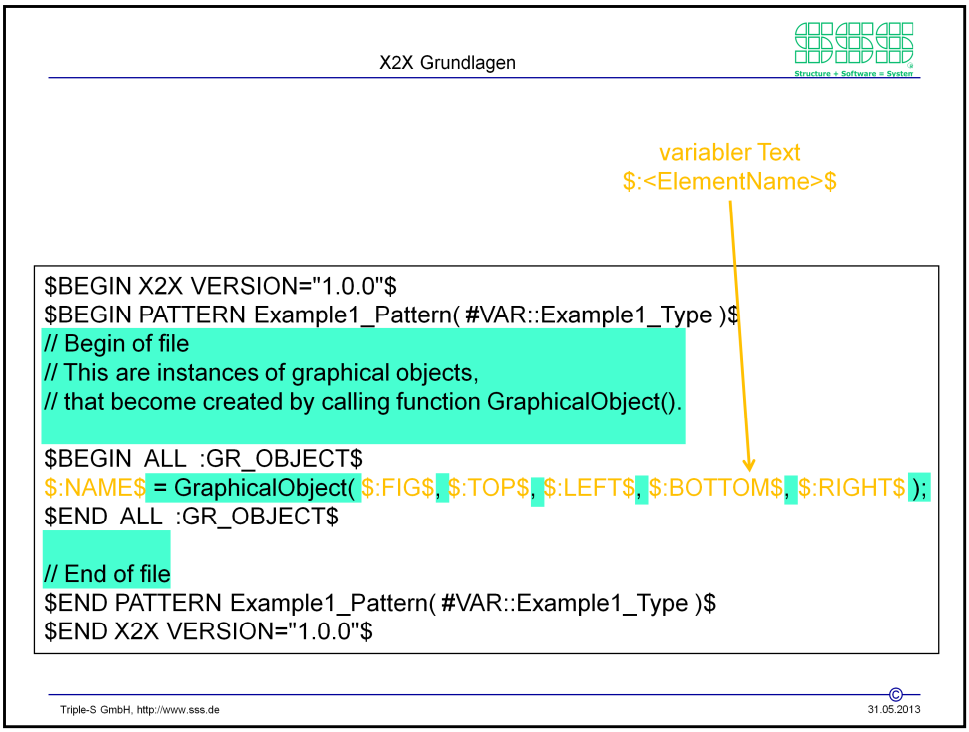
Dies entspricht noch nicht ganz unserer gewünschten Ausgabedatei.



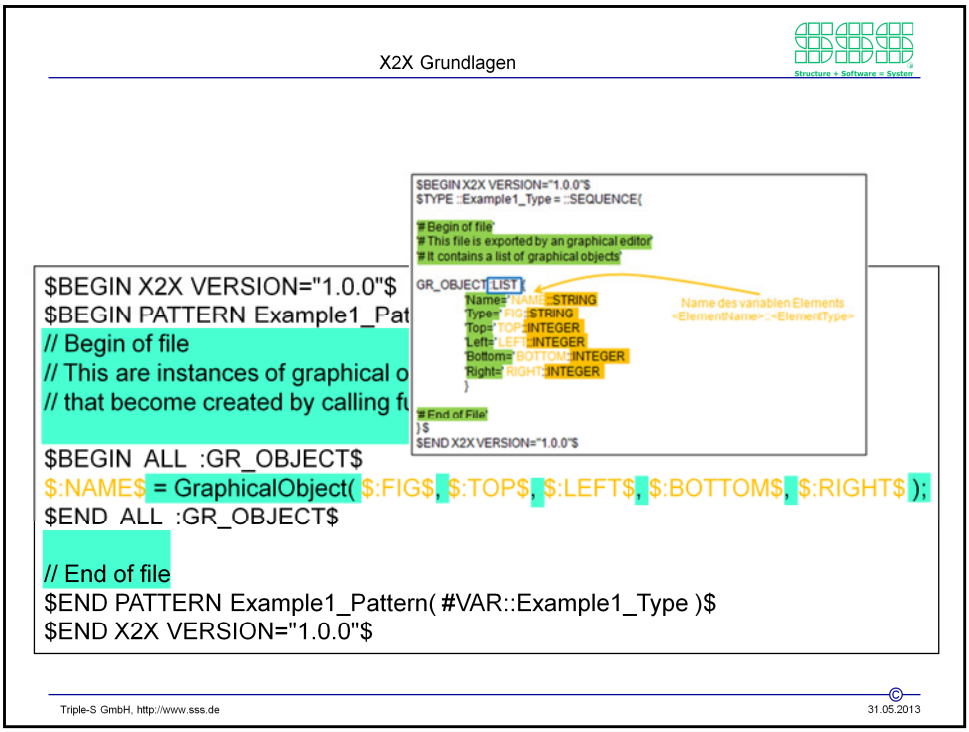
Zur Erinnerung:

Die Ausgabedatei enthält eine Liste mit konstantem und variablem Text.

Im Muster fehlen also noch die Werte der variablen Elemente. Diese erhalten wir, in dem wir dort die entsprechenden konstanten Werte durch Platzhalter ersetzen.



$:<ElementName>$ ist ein Platzhalter für ein variables Element und wird in der Ausgabedatei durch den tatsächlichen Wert des variablen Elements ersetzt.

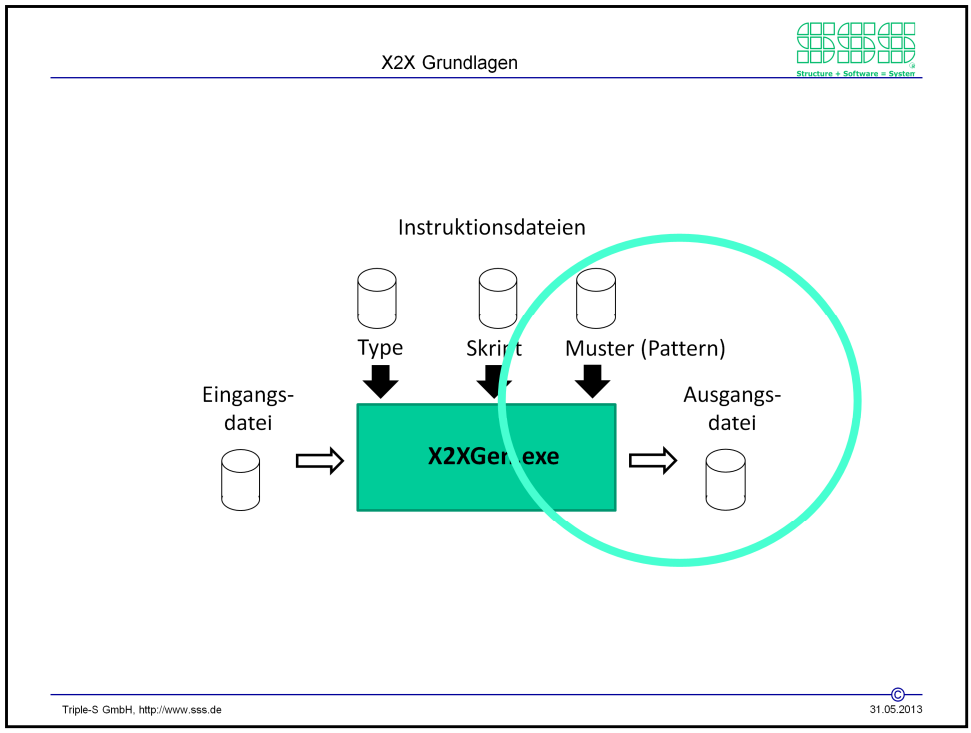


Die Namen der Platzhalter sind die Namen aus der Type-Datei!

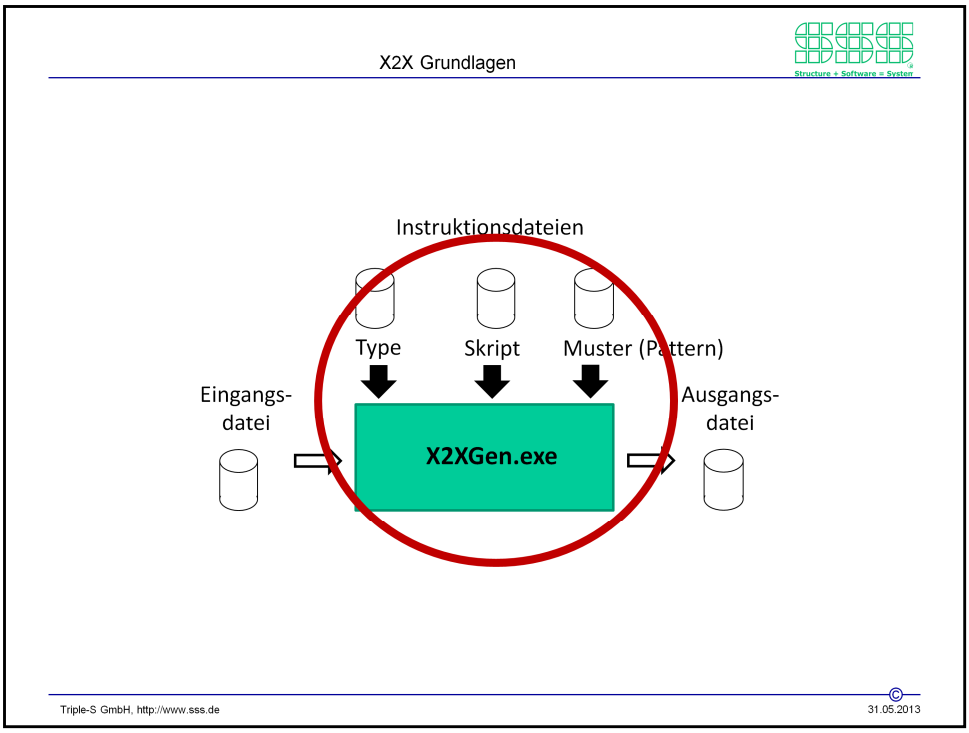
Fertig ist das Muster!

```
$BEGIN X2X VERSION="1.0.0"$
$BEGIN PATTERN Example1_Pattern( #VAR::Example1_Type )$
// Begin of file
// This are instances of graphical objects,
// that become created by calling function GraphicalObject().

$BEGIN  ALL  :GR_OBJECT$
$:NAME$ = GraphicalObject( $:FIG$, $:TOP$, $:LEFT$, $:BOTTOM$, $:RIGHT$ );
$END  ALL  :GR_OBJECT$

// End of file
$END PATTERN Example1_Pattern( #VAR::Example1_Type )$
$END X2X VERSION="1.0.0"$
```


Damit haben wir eine Muster-Datei erstellt, die genau unserer Ausgabedatei entspricht!

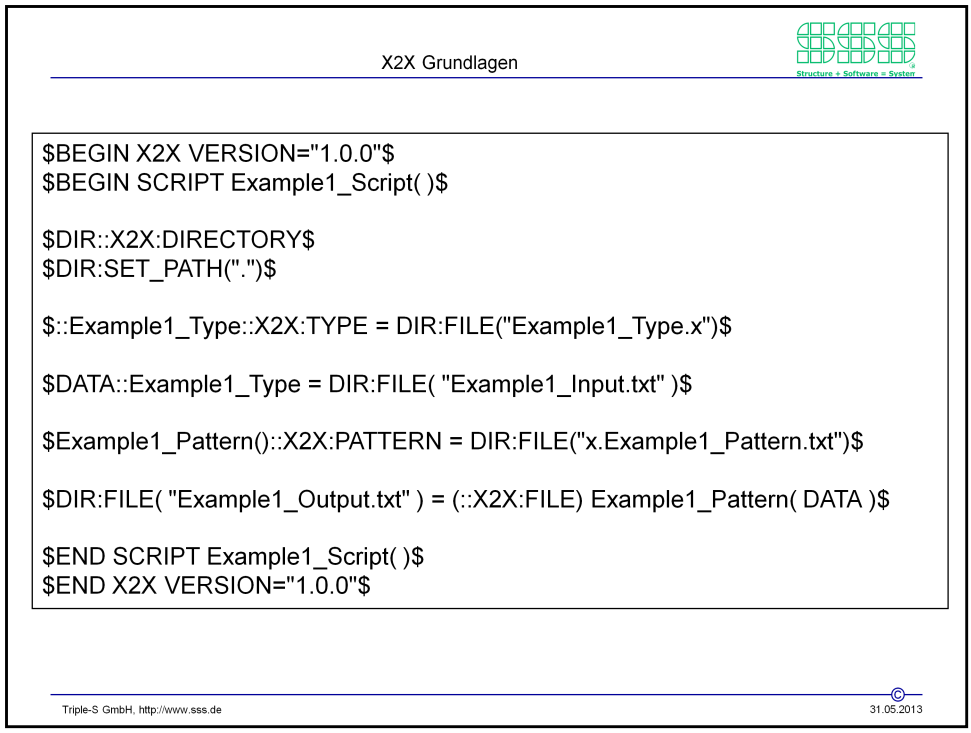


Jetzt sind sowohl der Type für die Eingabedatei als auch das Muster für die Ausgabedatei fertig gestellt.



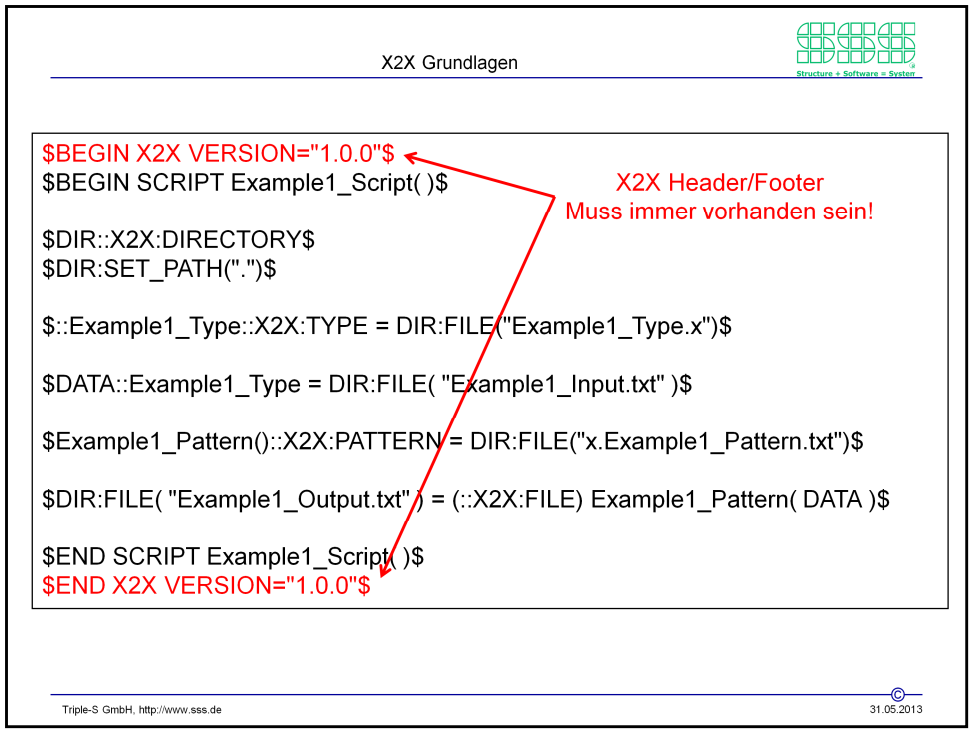
X2X braucht jetzt noch Anweisungen, welche konkrete Datei verwendet bzw. erstellt werden soll – also wie sie heißt und in welchem Verzeichnis sie liegt bzw. liegen soll.

Die Anweisungen werden in einem X2X Skript abgelegt.

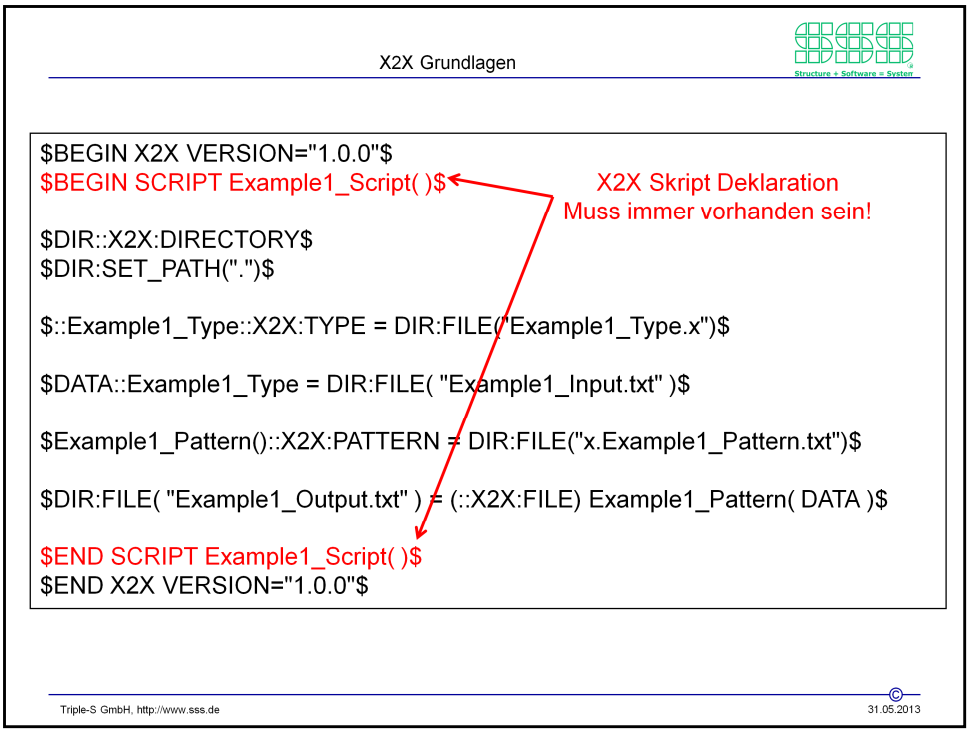


Dies ist das fertige Skript, das aus unserer Eingabedatei die Ausgabedatei erzeugt.

Wir gehen wieder nacheinander alle Anweisungen durch.



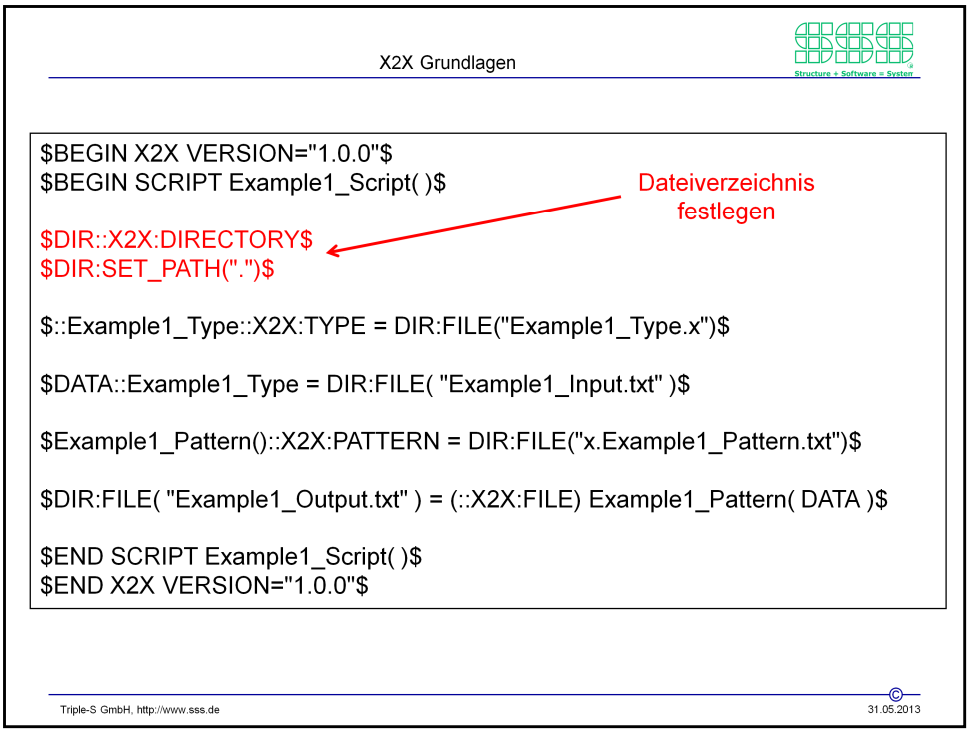
Wie bei Type und Muster muss auch in jedem Skript der X2X Header und Footer vorhanden sein.



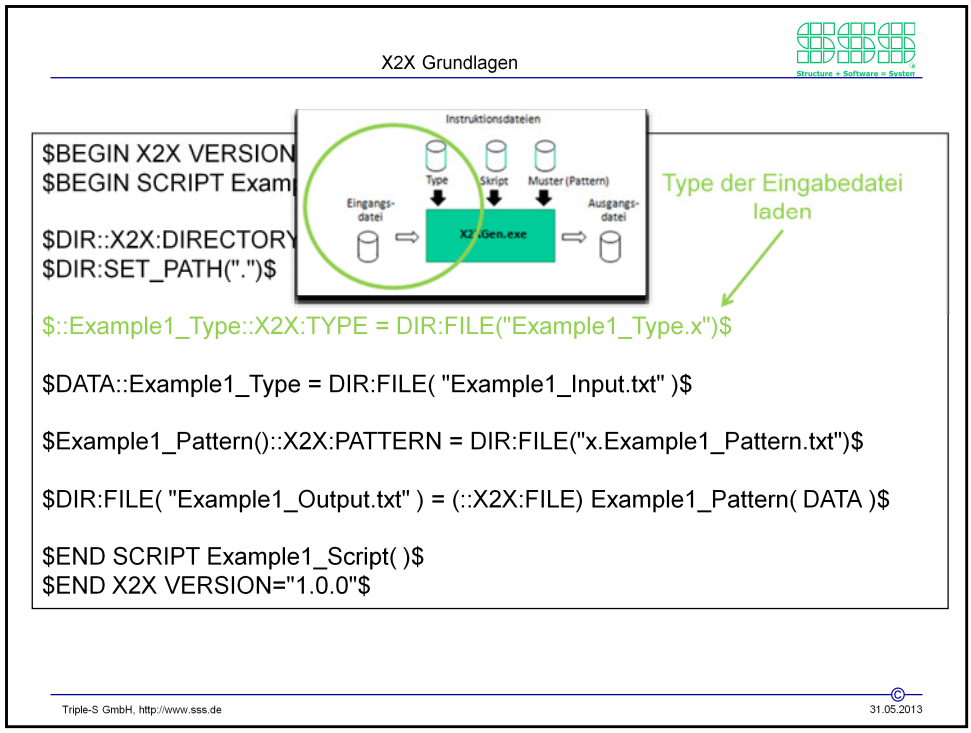
Es folgt die einleitende bzw. abschließende X2X Skript Deklaration als Blockanweisung.

Der entscheidende Unterschied zwischen ein Skript und einer Muster Deklaration ist das Sclüsselwort „SCRIPT“ bzw. „PATTERN“.

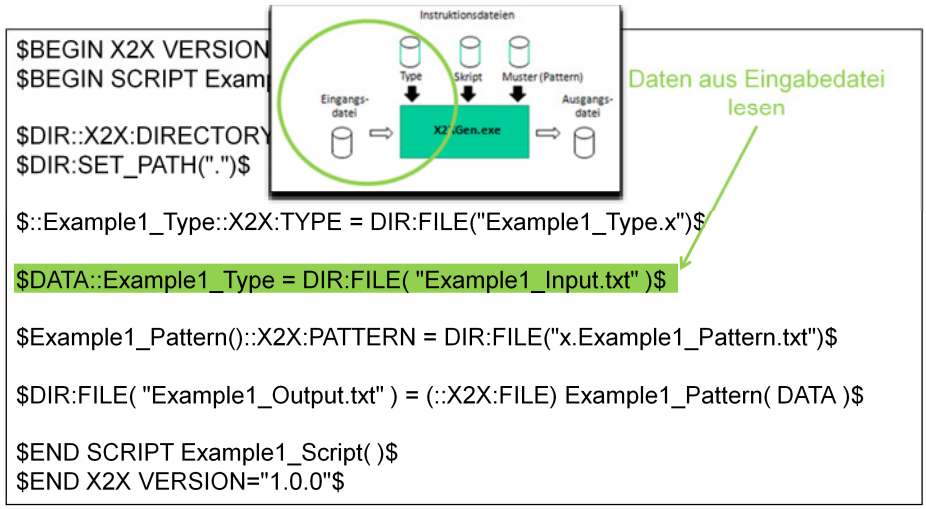
(PATTERN = englisch für MUSTER)



Diese beiden Anweisungen legen fest, dass DIR ein Dateiverzeichnis repräsentiert und auf das aktuelle Working-Verzeichnis verweist, in dem sich dieses Skript befindet.



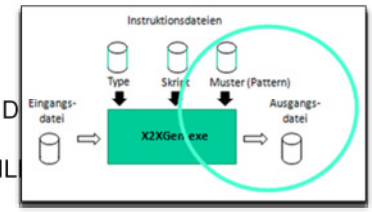
Der Type für unsere Eingabedatei hat den Namen „Example1_Type“ und soll aus der Datei „Example1_Type.x“ geladen werden.



Die variablen Daten aus der Eingabedatei „Example1_Input.txt“ sollen in der Datenstruktur „DATA“ abgelegt werden, wobei zum Lesen der eben geladene Type „Example1_Type“ verwendet werden soll.

```
$BEGIN X2X VERSION="1.0.0"$
$BEGIN SCRIPT Example1_Script( )$

$DIR::X2X:DIRECTORY$
$DIR:SET_PATH(".")$

$::Example1_Type::X2X:TYPE = D

$DATA::Example1_Type = DIR:FIL

$Example1_Pattern()::X2X:PATTERN = DIR:FILE("x.Example1_Pattern.txt")$

$DIR:FILE( "Example1_Output.txt" ) = (::X2X:FILE) Example1_Pattern( DATA )$

$END SCRIPT Example1_Script( )$
$END X2X VERSION="1.0.0"$
```

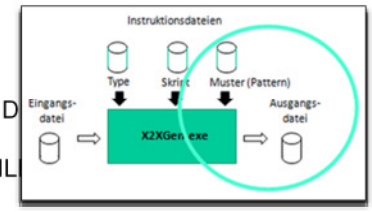
Muster der Ausgabedatei laden



Das Muster „Example1_Pattern“ für unsere Ausgabedatei soll aus der Datei „x.Example1_Pattern.txt“ geladen werden.

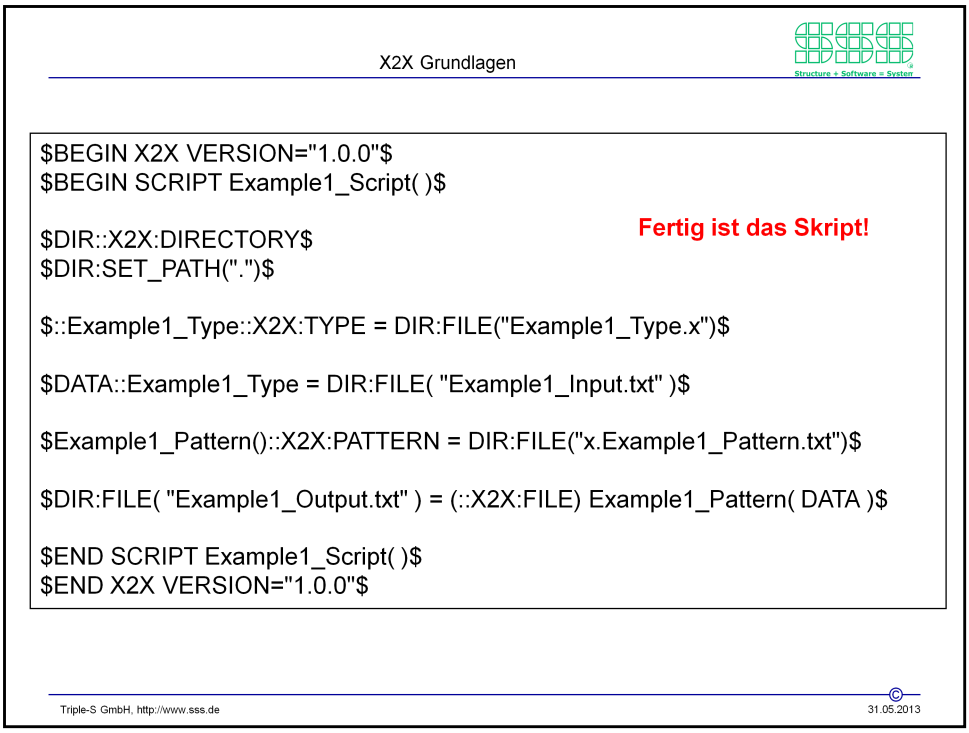
```
$BEGIN X2X VERSION="1.0.0"$
$BEGIN SCRIPT Example1_Script( )$

$DIR::X2X:DIRECTORY$
$DIR:SET_PATH(".")$

$::Example1_Type::X2X:TYPE = D

$DATA::Example1_Type = DIR:FIL

$Example1_Pattern()::X2X:PATTERN = DIR:FILE("x.Example1_Pattern.txt")$

$DIR:FILE( "Example1_Output.txt" ) = (::X2X:FILE) Example1_Pattern( DATA )$

$END SCRIPT Example1_Script( )$
$END X2X VERSION="1.0.0"$
```

Ausgabedatei aus Muster mit gelesenen Daten erzeugen

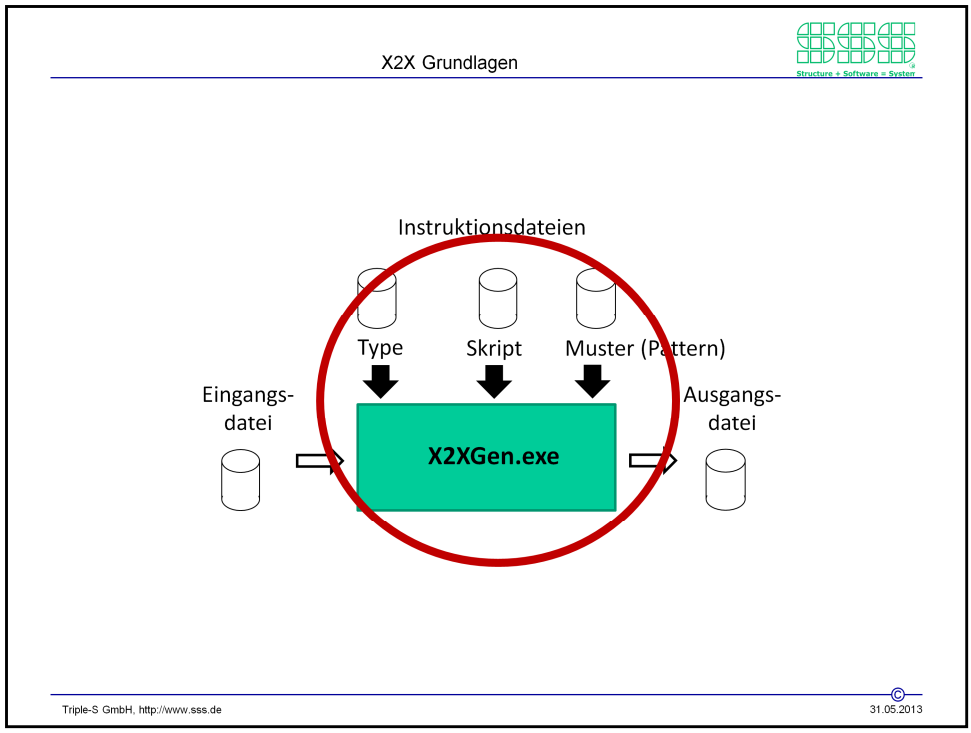


Unsere Ausgabedatei soll mit dem Namen „Example1_Output.txt“ erstellt werden.

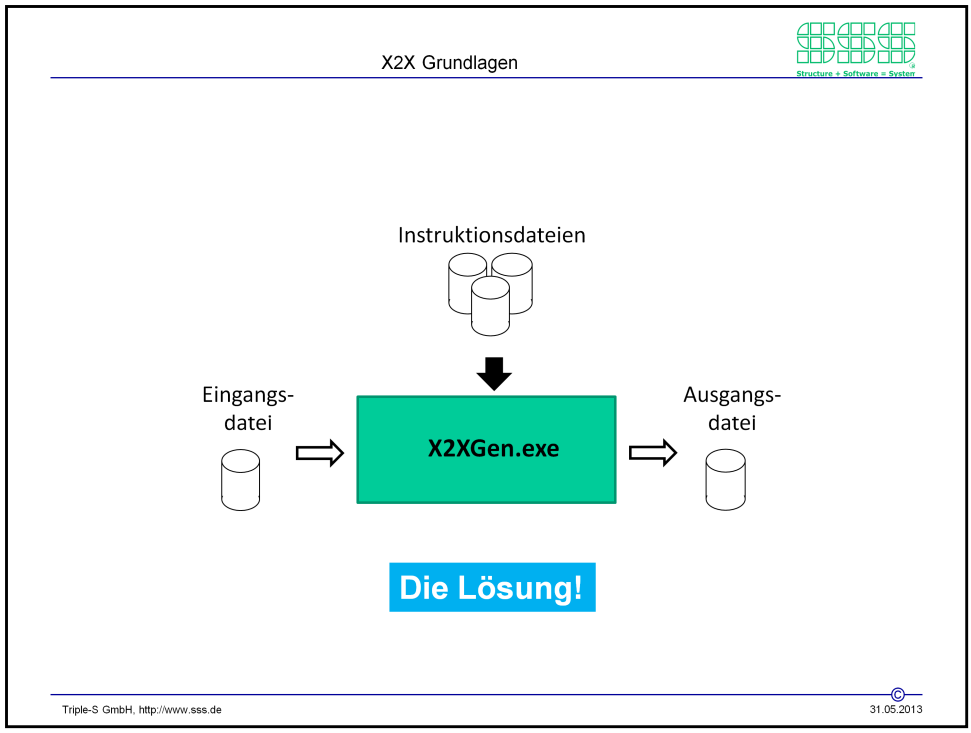
Dabei soll das Muster „Example1_Pattern“ verwendet werden, wobei die Werte der variablen Elemente in der Datenstruktur „DATA“ zu finden sind.



Fertig ist unser Skript. X2X weiß jetzt genau, was zu erledigen ist.

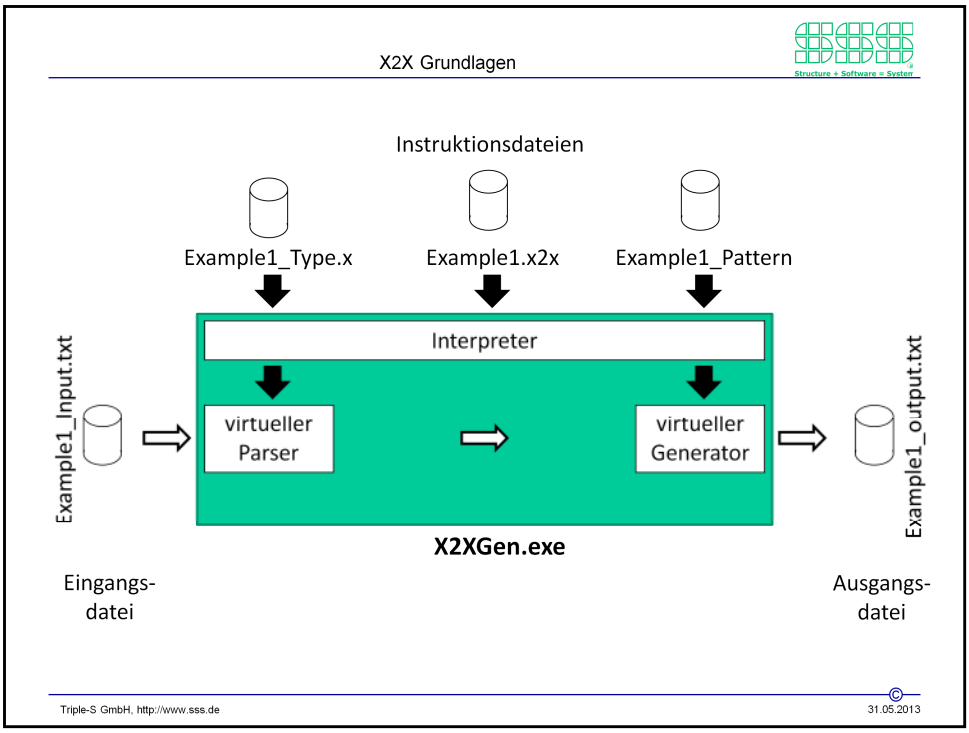


Mit dem X2X Skript haben wir den letzten Baustein erstellt, den X2X braucht um die Aufgabe zukünftig auf Knopfdruck für uns zu erledigen.



Jetzt sind Ihnen die wesentlichen Dinge zu X2X bekannt!

War doch gar nicht so schlimm! – Oder?

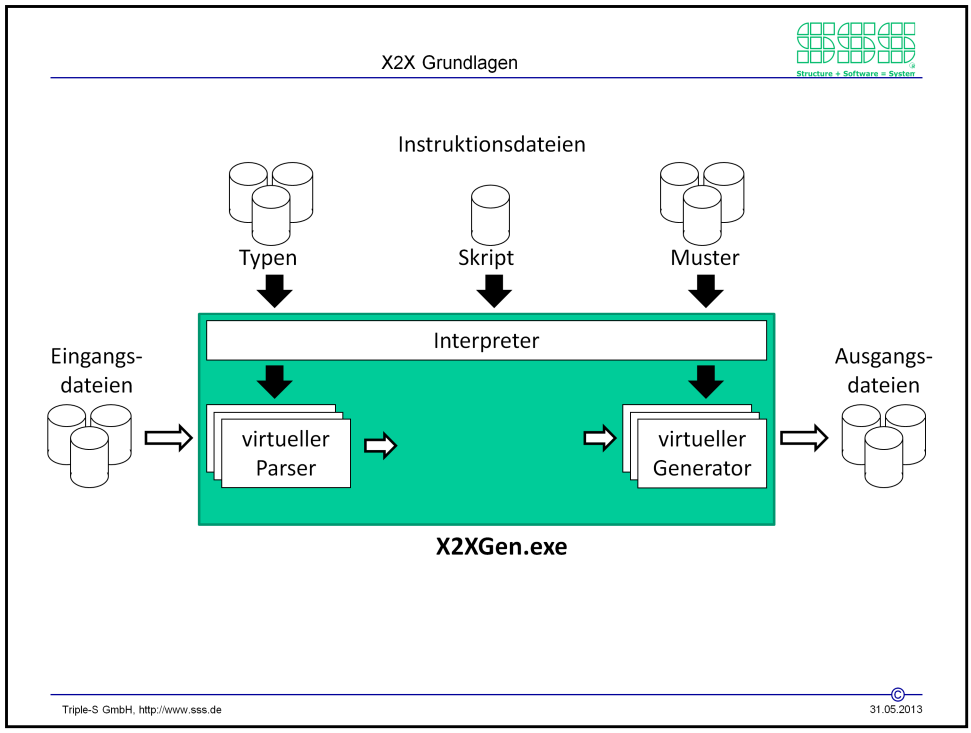


Eine letze Anmerkung:

Unser Beispiel war insofern einfach, als wir nur eine Eingabedatei haben und daraus nur eine Ausgabedatei erstellen müssen.

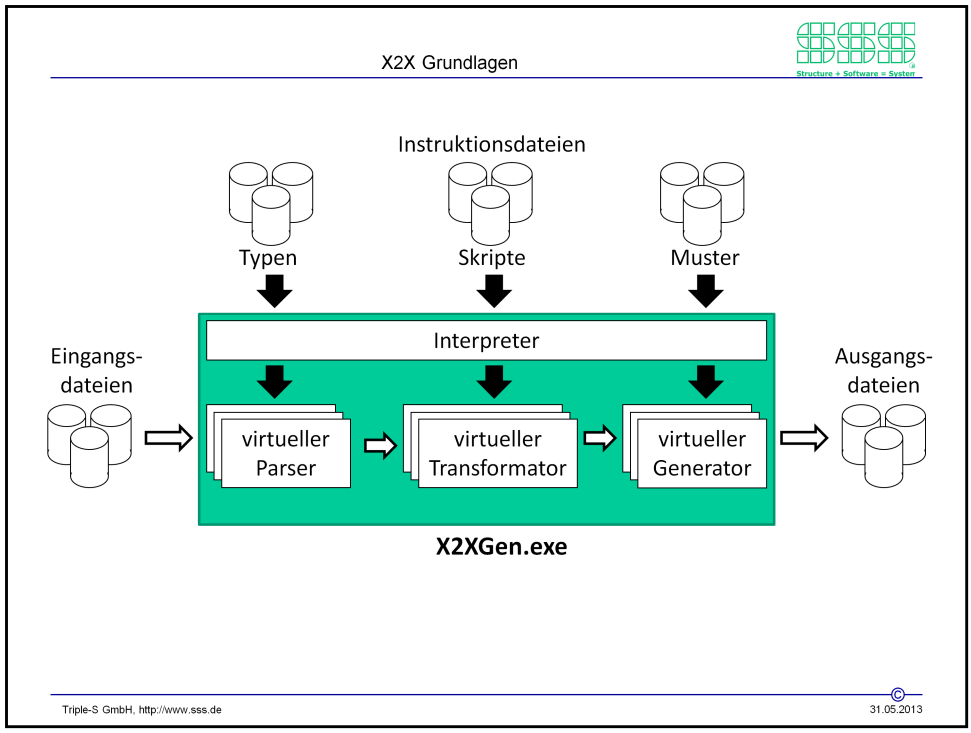
Dies ist zwar ein in der Praxis häufig auftretender Fall, aber genau so häufig kommt es in der Praxis vor, dass die Daten mehrerer verschiedenartig strukturierter Eingabedateien miteinander kombiniert und in einer oder mehreren Ausgabedateien neu formatiert abgelegt werden müssen.

Die zugehörigen X2X Skripte und Muster sind deshalb aber nicht wesentlich komplizierter. Es müssen lediglich mehrere Typen und Muster gleichzeitig geladen werden und Muster haben eventuell mehrere Datenstrukturen als Parameter.



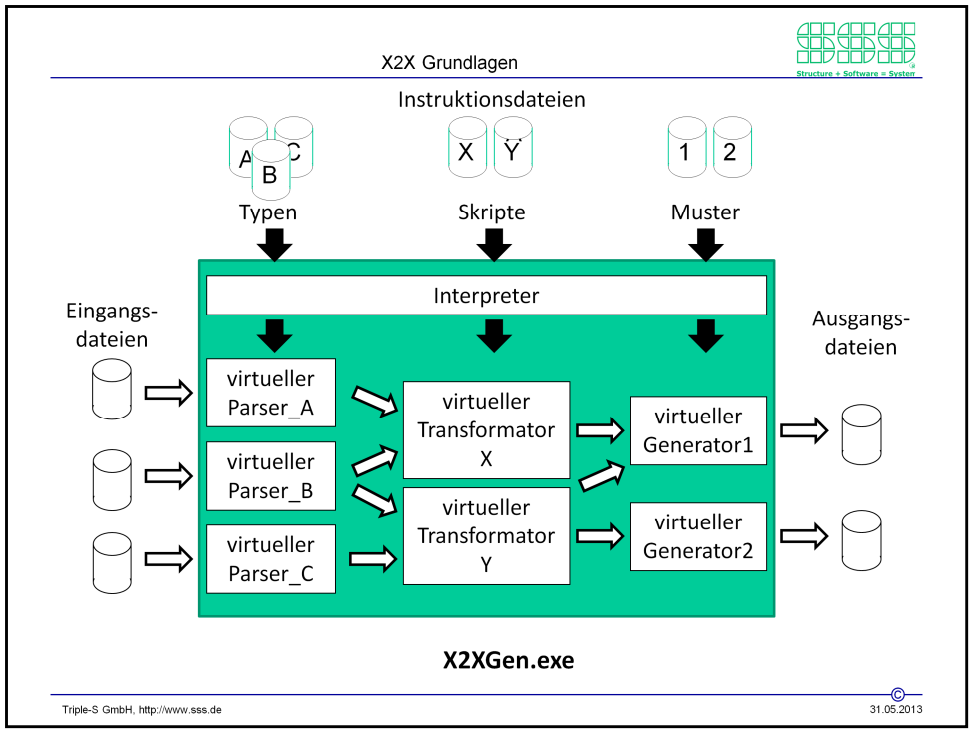
Unser Beispiel war auch insofern einfach, als wir die variablen Werte aus der Eingabedatei nur unverändert in der Ausgabedatei einsetzten.

Dies ist zwar ein in der Praxis häufig auftretender Fall, aber genau so häufig müssen die variablen Ausgabewerte vorher erst durch Kombination oder Berechnung aus den Eingabewerten ermittelt werden. Dazu kann man in einem Skript oder Muster Variablen anlegen und z.B. numerische Berechnungen damit durchführen.

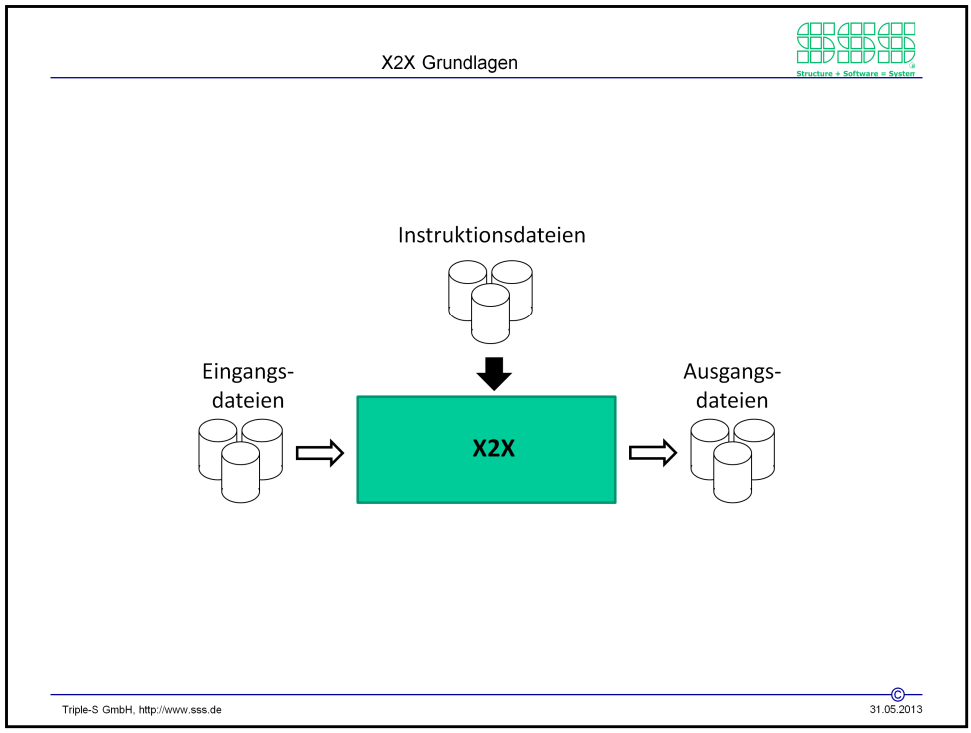


Zwischen dem reinen Einlesen der Daten und dem reinen Ausgeben von Daten kann also noch eine Datentransformation platziert werden.

Sollen bestimme Datentransformationen wiederverwendet werden, kann man sie in Skript-Makros bzw. Muster-Makros ablegen. Diese sind ganz genauso wie normale Skripte bzw. Muster aufgebaut.

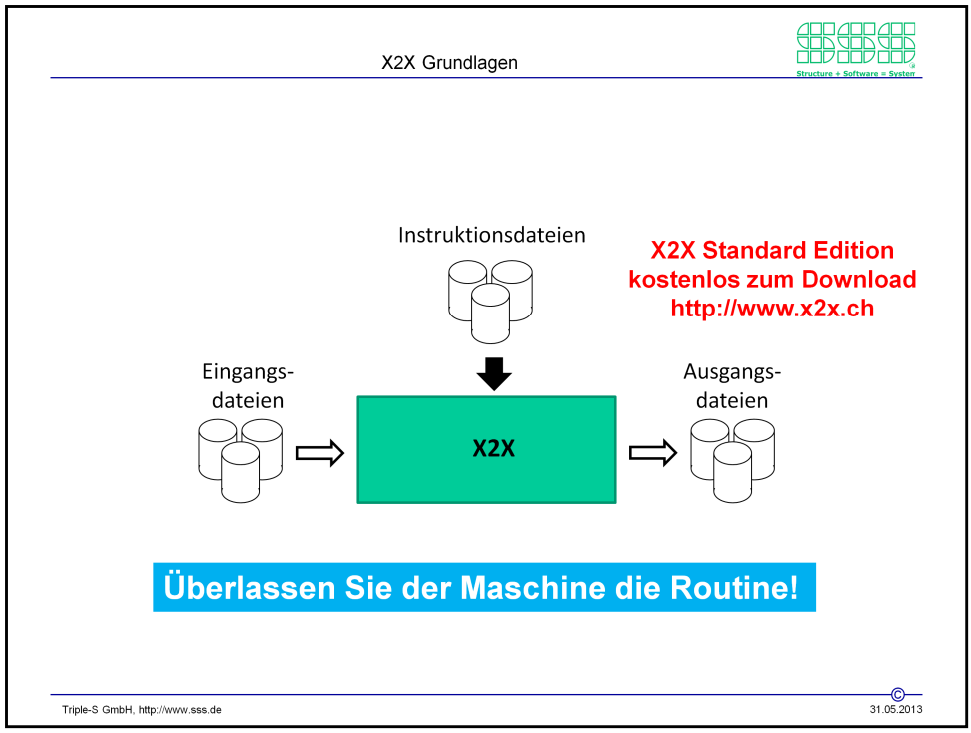


X2X intern kann das ganz schön kompliziert werden, was X2X da leisten muss …



… aber keine Sorge, damit haben Sie nichts zu tun.

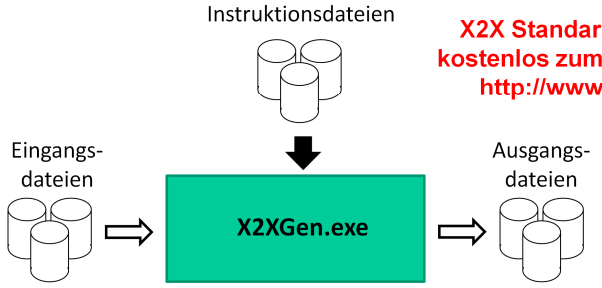
X2X managed das alles für Sie!



Sicherlich haben sie jetzt Lust bekommen X2X gleich auszuprobieren …?

Eine Vollversion zum Testen von X2X gibt es unter http://www.x2x.ch zum download.

Vielen Dank für Ihre Aufmerksamkeit



X2X Standard Edition
kostenlos zum Download
http://www.x2x.ch

Überlassen Sie der Maschine die Routine!

Vielen Dank für Ihre Aufmerksamkeit … und viel Spaß mit X2X!