

Diplomarbeit

Titel der Diplomarbeit:

BNF4OOo - Managing Backus-Naur-Forms with OpenOffice

Verfasserin/Verfasser:

Franz Hohenegger

Matrikel-Nr.:

0151047

Studienrichtung:

Betriebswirtschaft

Beurteilerin/Beurteiler:

Prof. Dr. Rony G. Flatscher

Ich versichere, dass:

ich die Diplomarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.

ich dieses Diplomarbeitsthema bisher weder im In- noch im Ausland (einer Beurteilerin/einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

diese Arbeit mit der vom Begutachter/von der Begutachterin beurteilten Arbeit übereinstimmt.

Datum

Unterschrift

Abstract

This diploma thesis is about a program named BNF4OOo which enables the user to manage different supported BNF-dialects and make own customized BNF-dialects as well. These dialects can be transferred into character-based syntax diagrams and a XML-format closely related to IBM's DITA. OpenOffice is used as a graphical interface for the transformations. This work describes the supported BNF-dialects with their used syntax structures and their implementation in BNF4OOo. Furthermore it introduces the internal logic and structure of BNF4OOo.

Thanks

I want to thank Prof. Dr. Rony G. Flatscher for his great help and support during the creation of this paper. Without his impulses BNF4OOo would not have this set of functions. Furthermore BNF4OOo would not run without his work on BSF4ooRexx and log4rex. In addition I want to thank Jean-Louis Faucher for showing me IBM's DITA and giving me important information for improving the BNF-parser.

Table of Contents

1	Introduction.....	10
1.1	Requirements.....	11
1.2	Used Environments and Version.....	11
1.3	Roadmap.....	12
2	BNF4OOo.....	13
2.1	Supported BNF-Dialects.....	13
2.1.1	Backus-Naur-Form.....	13
2.1.2	Common Variations of BNF.....	14
2.1.3	XML-EBNF.....	15
2.1.3.1	BNF Structures.....	15
2.1.3.2	New Structures.....	15
2.1.3.3	New Expressions.....	16
2.1.4	ISO-EBNF.....	16
2.1.4.1	Defining-symbol.....	17
2.1.4.2	Definition-separator-symbol.....	17
2.1.4.3	Concatenate-symbol.....	17
2.1.4.4	Terminator-symbol.....	17
2.1.4.5	Grouped-Sequence.....	17
2.1.4.6	Optional-Sequence.....	18
2.1.4.7	Repeated-sequence.....	18
2.1.4.8	Syntactic-factor.....	18
2.1.4.9	Quote-symbols.....	18
2.1.4.10	Syntactic-exception-symbol.....	18
2.1.4.11	Comment-symbols.....	18
2.1.4.12	Special-sequence-symbols.....	18
2.1.5	ABNF.....	19
2.1.5.1	Terminal Values.....	19
2.1.5.2	Rule Form.....	20
2.1.5.3	Concatenation.....	20
2.1.5.4	Alternatives.....	20
2.1.5.5	Incremental Alternatives.....	20
2.1.5.6	Value Range Alternatives.....	21

2.1.5.7 Sequence Group.....	21
2.1.5.8 Variable Repetition.....	21
2.1.5.9 Specific Repetition.....	22
2.1.5.10 Optional Sequence.....	22
2.1.5.11 Comment.....	23
2.1.6 Not Supported by BNF4OOo, xBNF.....	23
2.2 Positions of BNF-symbols.....	23
2.3 Supported BNF-Instructions.....	24
2.3.1 Definition.....	25
2.3.2 Integrated-Definition.....	26
2.3.3 XOR-Definition.....	27
2.3.4 Delimiter.....	28
2.3.5 Terminator.....	29
2.3.6 XOR.....	29
2.3.7 Default.....	31
2.3.8 Comment.....	31
2.3.9 Nonterminal.....	32
2.3.10 Literal.....	33
2.3.11 Range.....	33
2.3.12 Group.....	34
2.3.13 Optional.....	34
2.3.14 Loop.....	35
2.3.15 Optional-loop.....	35
2.3.16 Argument.....	36
2.3.17 Factor.....	37
2.3.18 Complex-factor.....	37
2.3.19 Exception.....	38
2.3.20 Overview on the Supported BNF-Instructions.....	38
2.4 The Syntax Diagram Types.....	39
2.4.1 ASCII Image.....	42
2.4.2 Unicode Image.....	42
2.4.3 Boxed Unicode Image.....	43
2.5 BNF4OOo XML Format.....	44
2.6 The BNF4OOo GUI.....	50

2.6.1	BNF2Diagram.....	50
2.6.2	BNF2XML.....	50
2.6.3	XML2BNF	51
2.6.4	BNF2All.....	51
2.6.5	Import Data.....	53
2.6.6	Export Diagram.....	54
2.6.7	Options.....	55
2.7	BNF4OOo OptionGUI.....	55
2.7.1	General Options.....	57
2.7.2	Appearance Options.....	59
2.7.3	BNF-dialects.....	61
2.7.4	Drawing-symbols.....	63
2.7.5	Export Options.....	65
2.8	BNF4Shell.....	68
3	BNF4OOo's Internal Structure and Logic.....	70
3.1	Loading the BNF-dialects.....	71
3.2	The Input and Output System.....	75
3.3	Generating the Object Model.....	78
4	Outlook.....	82
5	References.....	83
6	Attachment.....	88
6.1	Installation Guide.....	88
6.1.1	Downloading the Components.....	88
6.1.2	Running the Installation Script.....	88
6.1.3	Integrating the Startscript in the OOo Toolbar.....	90
6.2	The OptionGUI – A Quick Overview.....	94
6.3	The BNF4OOo Files – A Quick Overview.....	95

List of BNFs

BNF 1:	Example for a BNF-rule.....	14
BNF 2:	Example for Terminals and Nonterminals.....	14
BNF 3:	XOR example.....	14
BNF 4:	ISO/IEC Concatenate-symbol example.....	17
BNF 5:	ISO/IEC repetition-symbol example.....	18

BNF 6: ABNF notation for alternatives.....	20
BNF 7: ABNF notation for incremental alternatives.....	20
BNF 8: ABNF notation mix of alternatives and incremental alternatives.....	21
BNF 9: ABNF value range alternatives example.....	21
BNF 10: ABNF variable repetitions example.....	21
BNF 11: ABNF specific repetitions example 1.....	22
BNF 12: ABNF specific repetitions example 2.....	22
BNF 13: Suffix notation example.....	23
BNF 14: Circumfix notation example.....	24
BNF 15: Definition-instruction with comment example.....	25
BNF 16: Definition-instruction without comment example.....	25
BNF 17: Splitted BNF-rules example 1.....	25
BNF 18: Splitted BNF-rules example 2.....	26
BNF 19: two related BNF-rules.....	26
BNF 20: XOR-definition example 1.....	27
BNF 21: XOR-definition example 2.....	28
BNF 22: delimiter-instruction example.....	28
BNF 23: Terminator-instruction example.....	29
BNF 24: XOR-instruction example 1.....	30
BNF 25: XOR-instruction example 2.....	30
BNF 26: XOR-instruction example 3.....	30
BNF 27: Default-instruction example.....	31
BNF 28: Comment-instruction example.....	32
BNF 29: Nonterminal-instruction example.....	33
BNF 30: Range-instruction example.....	33
BNF 31: Group-instruction example.....	34
BNF 32: Optional-instruction example.....	34
BNF 33: Loop-instruction example.....	35
BNF 34: Optional-loop-instruction example.....	35
BNF 35: Argument-instruction example 1.....	36
BNF 36: Argument-instruction example 2.....	36
BNF 37: Factor-instruction example.....	37
BNF 38: Complex-factor-instruction example.....	37
BNF 39: Exception-instruction example.....	38

BNF 40: Command strings in streaming functions.....	41
BNF 41: Illustration of the BNF4OOo XML format.....	45
BNF 42: Two unrelated BNF rules in one XML file.....	46
BNF 43: Example for groupchoices and optional sequences.....	47
BNF 44: Example for loops in DITA.....	47
BNF 45: Example for exceptions – not supported by DITA.....	48
BNF 46: Example for fragments.....	49
BNF 47: A BNF-rule containing a XOR.....	72
BNF 48: Two unrelated BNF-rules.....	78

List of Codes

Code 1: Methods of the .Instruction class.....	72
Code 2: The .Instruction_group class.....	73
Code 3: The .BNFInstructionParser's end_element method.....	75
Code 4: Creating a new_instruction object in .BNFInstructionparser.....	75
Code 5: Generating the object modell in XML2BNF.....	78
Code 6: The .BNFClass.....	80

List of Figures

Figure 1: Table of ISO/IEC EBNF structures.....	19
Figure 2: ABNF character notation example.....	20
Figure 3: Comparison of BNF, EBNF and XBNF syntax.....	23
Figure 4: Overview of BNF-dialects and DITA.....	39
Figure 5: Streaming syntax diagram.....	40
Figure 6: BNF4OOo Toolbar.....	50
Figure 7: BNF2All-part1.....	52
Figure 8: BNF2All-part2.....	53
Figure 9: Import data content dialog window.....	54
Figure 10: Import data dialog window.....	54
Figure 11: OptionGUI properties.....	56
Figure 12: OptionGUI menu bar.....	57
Figure 13: General Options.....	58
Figure 14: Appearance Options.....	60
Figure 15: BNF-Dialects.....	62

Figure 16: Drawing-symbols.....	64
Figure 17: Allowed UTF-8 codings.....	65
Figure 18: Export Options.....	66
Figure 19: BNF4Shell.rex options.....	68
Figure 20: BNF4OOo function's workflow.....	70
Figure 21: .Instruction and its subclasses.....	74
Figure 22: Schematic overview of BNF4OOo and BNF4Shell.rex.....	76
Figure 23: Important input and output methods of OOoInterface.rex.....	77
Figure 24: Important input and output methods of OOoInterface.rex.....	77
Figure 25: The object tree generated out of BNF 48.....	79
Figure 26: .BNFClass and its subclasses.....	81
Figure 27: Running addBNF4OOoButtons.rex as a macro part 1.....	89
Figure 28: Running addBNF4OOoButtons.rex as a macro part 2.....	89
Figure 29: Entering the toolbar customize menu.....	90
Figure 30: Creating a new menu part 1.....	91
Figure 31: Creating a new menu part 2.....	91
Figure 32: Adding Entries to the new menu.....	92
Figure 33: Adding addBNF4OOoButtons.rex to the menu.....	92
Figure 34: Running addBNF4OOoButtons.rex from the toolbar.....	93
Figure 35: Removing BNF4OOo from the toolbar.....	93
Figure 36: Quick overview on the BNF4OOo subfolders.....	94
Figure 37: Quick overview on the BNF4OOo subfolders.....	95
Figure 38: Quick overview on the BNF4OOo files part 1.....	96
Figure 39: Quick overview on the BNF4OOo files part 2.....	97

List of Syntax Diagrams

SD 1: ABNF specific repetitions example 1.....	22
SD 2: ABNF specific repetitions example 2.....	22
SD 3: First SD created from BNF 19 when using a definition-instruction.....	26
SD 4: Second SD created from BNF 19 when using a definition-instruction.....	26
SD 5: Integrated-definition example 3.....	27
SD 6: XOR-definition example 1.....	28
SD 7: XOR-definition example 2.....	28
SD 8: delimiter-instruction example.....	28

SD 9: Terminator-instruction example part 1.....	29
SD 10: Terminator-instruction example part 2.....	29
SD 11: XOR-instruction example 1.....	30
SD 12: XOR-instruction example 2.....	30
SD 13: XOR-instruction example 3.....	31
SD 14: Default-instruction example.....	31
SD 15: Comment-instruction example part 1.....	32
SD 16: Comment-instruction example part 2.....	32
SD 17: Nonterminal-instruction example.....	33
SD 18: Range-instruction example.....	33
SD 19: Group-instruction example.....	34
SD 20: Optional-instruction example.....	34
SD 21: Loop-instruction example.....	35
SD 22: Optional-loop-instruction example.....	35
SD 23: Argument-instruction example 1.....	36
SD 24: Argument-instruction example 2.....	36
SD 25: Factor-instruction example.....	37
SD 26: Complex-factor-instruction example.....	38
SD 27: Exception-instruction example.....	38
SD 28: Command strings in streaming functions – ASCII image.....	42
SD 29: Command strings in streaming functions – Unicode image.....	43
SD 30: Command strings in streaming functions – boxed-Unicode image.....	44
SD 31: Example for fragments part 1.....	49
SD 32: Example for fragments part 2.....	49
SD 33: Example for fragments part 3.....	50
SD 34: Diagram with a minimum length set to 0.....	61
SD 35: Diagram with minimum length set to 81.....	61
SD 36: The diagram generated from BNF 47 with long=true.....	72
SD 37: The diagram generated from BNF 47 without long=true.....	72

List of XMLs

XML 1: Illustration of the BNF4OOo XML format.....	45
XML 2: Two unrelated BNF rules in one XML file.....	46
XML 3: Example for groupchoices and optional sequences.....	47

XML 4: Example for loops in DITA.....	48
XML 5: Example for exceptions – not supported by DITA.....	48
XML 6: Example for fragments.....	49
XML 7: BNF-ALGOL_60.XML.....	71

1 Introduction

This diploma thesis is about a program named BNF4OOo. The purpose of this program is to generate syntax diagrams out of BNF-dialects inside an OpenOffice `sWriter` document.

In every developing process for programming languages and frameworks programmers have to deal with the topic of documentation. An only verbal description of the syntax structures consumes a lot of time not only for the author but also for the reader. Therefore the metasyntax BNF was created to provide a compact and simple way to describe the syntax of programming languages. Over the years BNF was altered, enhanced and extended into different BNF-dialects in order to fit different purposes. Therefore, developers who want to use this approach nowadays are confronted with the problem of finding a BNF-dialect which fits best for their demands.

Another approach based on BNF is to describe the syntax not with text but with syntax diagrams. The advantage is that these diagrams are more readable with a clearer meaning compared to BNF-dialects but on the other hand drawing these diagrams by hand is a time consuming process as well. Generating these diagrams automatically is the obvious solution but shifts the problem back to the choice of the underlying BNF-dialect. Syntax diagram generating programs usually concentrate on the syntactic elements of one or a few BNF-dialects.¹

BNF4OOo takes another approach. It aims to support nearly all syntactic elements from the most known BNF-dialects but to separate them from their actual symbol (e.g. `|` for representing alternatives). This way not only all available BNF-dialects can be used, it is also possible to create own customized BNF-dialects. In addition BNF4OOo uses OpenOffice as a graphical interface so the generated diagrams can be generated directly into the documentation and the whole functionality of OpenOffice is available for further processing (e.g. the PDF export).

Generating syntax diagrams for a documentation is not a everyday task but is usually done one time when releasing a new version. Since this release circles can take years, BNF4OOo aims to take a minimum effort for programmers to actualize their documentations.

¹ C.f [BrauFra], [RailRoad], [SchoDo], [ThiePe]

1.1 Requirements

BNF4OOo is not a stand-alone piece of software. In order to get BNF4OOo working, some programs have to be installed first. Each of these programs is available for free on the Internet for Windows, Mac and Linux:

- ooRexx ,
- BSF4ooRexx,
- Java SDK,
- OpenOffice.

In addition, the following external ooRexx scripts are already embedded in the BNF4OOo program:

- log4rex,
- bsfScreenShot.cls,
- xmlparser.cls.

1.2 Used Environments and Version

For creating BNF4OOo the following programs where used (in alphabetical order):

- BSF4ooRexx 4.06, released on 27.03.2011² (contains bsfScreenShot.cls),
- Eclipse Classic 3.6.2³,
- Java JDK 1.6.0_24⁴,
- log4rex⁵,
- OpenOffice.org 3.2.1⁶ ,
- ooRexx 4.1.0⁷,
- RexxEclipse 0.1.1.1 ⁸.

² C.f. [BSF4Re]

³ C.f. [Eclipse]

⁴ C.f. [JavaJDK]

⁵ C.f. [log4rex]

⁶ C.f. [OOo]

⁷ C.f. [ooRexx]

⁸ C.f. [KriTam]

- `xmlparser.cls`⁹

1.3 Roadmap

The first chapter describes why BNF4OOo was programmed, what pieces of software are required to use it and which programs and environments were used to create the software itself and this diploma thesis. Chapter two gives an overview on the most common known BNF-dialects, their symbols and their implementation in BNF4OOo and the BNF4OOo functions to transform BNFs into XML files, XML files into BNFs and BNFs into syntax diagrams. Finally, the third chapter takes a look inside the internal structures of BNF4OOo with some hints how to enhance the program.

⁹ C.f. [XMLPar]

2 BNF4OOo

The following chapters give a brief overview on the most common known BNF-dialects and their implementation in BNF4OOo. The concept of railroad diagrams is introduced in context with the BNF4OOo's character based drawing approach and BNF4OOo's XML format based on IBM's DITA is described.

2.1 Supported BNF-Dialects

Since the Backus-Naur-Form (BNF) was invented to formally describe the syntax of the programming language ALGOL 58, it has been altered and extended from different organizations over the last decades. [Wikiped01].

Since most variations only differ by a different notation of the elements by using other strings and/or other positioning for the elements, the number of variations is overwhelming. One reason for this spreading is that the describing metalanguage should not use keywords of the programming language which is described. [ISOEBNF01]. Although there are ways to handle this problem (e.g. by quoting the keywords) it makes the written lines a bit harder to read for the human eye. Some approaches intended to find a new general standard while others took a more pragmatic way by defining BNF-dialects for their own actual needs only.

The BNF4OOo approach does not try to make another general standard, but to provide a framework which enables the user to create his own solution. It implements most of the general standards and allows the user to alter and extend them. The explicitly supported BNF-dialects are listed and explained briefly in the following chapters. In these explanations, the used structures are named with the words used in their documentations. Sometimes these names might appear confusing (e.g. concatenate-symbol is used in ISO EBNF to describe a `delimiter` symbol). Therefore their BNF4OOo counterpart will be mentioned as well. Only their used characters (e.g. `::=`) and structures with a name that matches their BNF4OOo counterpart are highlighted (e.g. `definition`)

2.1.1 Backus-Naur-Form

John Backus, a programming language designer used a context free grammar to describe the syntax of ALGOL 58 in 1959. In 1963 it was revised and expanded for ALGOL 60 by Peter Naur who named Backus notation Backus Normal Form. Later on, due to Donald Knuth's argumentation that it is not a "normal form in the conventional sense" it was renamed Backus Naur Form. [Knuth], [Wikiped01], [Wikiped02].

A BNF specification is a set of rules with the syntax as described in BNF 1.

```
<defined_symbol> ::= definition_of_the_symbol
```

BNF 1: Example for a BNF-rule.

The symbol `::=` delimits the left side containing the symbol which needs to be defined from the right side which contains the defining elements. In BNF4OOo it is referred to as the `definition-symbol` (see 2.3.1, page 26). "Atomic-symbols", meaning that they are not defined in any other rule, are called `Terminals` whereas all other symbols are called `Nonterminal` and are nested inside less-than `<` and greater-than `>` characters.

```
<A> ::= B <C> <D>
<C> ::= E F
<D> ::= G
```

BNF 2: Example for `Terminals` and `Nonterminals`

If a `Nonterminal` is not defined by a single set of symbols but by more than one alternatives, then these alternatives are delimited by a vertical bar, `|`. These alternatives will be called `XOR` in the following chapters (see 2.3.6, page 31). BNF 3 declares that `<A>` is either `B` or `C <D>` or `E`.

```
<A> ::= B | C <D> | E
```

BNF 3: `XOR` example

All other symbols than `::=`, `<`, `>`, and `|` are termed items.

2.1.2 Common Variations of BNF

The following list shows a number of common variations of the BNF notation. [Wikiped01].

- Grouped items are enclosed by simple parantheses (eg. $\langle A \rangle ::= (B \ C) \mid D$).
- A simple equal character $=$ is used instead of $::=$.
- Items repeating one or more times are enclosed in curly brackets
(e.g. $\langle A \rangle ::= \{B\}$).
- Items repeating one or more times are suffixed with an asterisk
(e.g. $\langle A \rangle ::= B^*$).
- Items repeating one or more times are suffixed with a plus char (e.g. $\langle A \rangle ::= B^+$).
- Optional items are enclosed in square brackets (e.g. $\langle A \rangle ::= [B]$).
- **Terminals** are written in bold and **NonTerminals** in plain text (not supported by BNF400o).
- **Terminals** are enclosed in quotes or apostrophes (e.g. $\langle A \rangle ::= "B"$).
- BNF rules end with a **terminator-symbol** $.$ (e.g. $\langle A \rangle ::= B.$).

Some variations and enhancements became own standards and are briefly described in the following chapters.

2.1.3 XML-EBNF

For describing the syntax of XML the W3C invented a simple Enhanced Backus-Naur Form (EBNF). [W3CXML]. Since its original purpose was not to establish a new standard but only to describe XML, it has only a short documentation. The enhancements can be divided into two groups. First new structures, and second new expressions for matching strings of one or more characters.

2.1.3.1 BNF Structures

Definitions $::=$ and **XOR-symbols** $|$ are the same as in the original BNF. **Nonterminal** items are written with an initial capital letter.

2.1.3.2 New Structures

The W3C enhanced BNF with the following structures:

- Comments start with `/*` and end with `*/`,
- Items can be grouped by enclosing them in simple brackets `(,)`,
- Optional Items are suffixed with `?`,
- Items repeating 0 or more times are suffixed with `*`,
- Items repeating 1 or more times are suffixed with `+`,
- `A - B` means that any string is correct that matches A but not B.

XML-EBNF also uses the expressions `[WFC: ...]` (well-formedness constraint) and `[VC: ...]` (validity constraint) which can be seen as some kind of comments inside the BNF rules, but their utility is limited to the XML description and cannot be adapted to other areas. BNF4OOo does not support them.

2.1.3.3 New Expressions

W3C uses a set of expressions to define valid characters:

- `#xN` expresses a character defined in ISO/IEC 10646 with the hexadecimal integer N and the prefix `#x`.
- An enumeration of characters can be defined by using square brackets `[,]`. This can be done by writing the character itself or with his hexadecimal number (e.g. `[abc]` or `[#xN#xN#xN]`).
- With `-` enclosed by `[` and `]` a range of valid characters can be defined (e.g. `[a-zA-Z]` or `[#xN-#xN]`).
- With `^` enclosed by `[` and `]` a range or enumeration of characters which are not valid can be defined (e.g. `[^a-z]`, `[^#xN-#xN]`, `[^abc]` or `[^#xN#xN#xN]`).
- Literal strings are enclosed by quotes `"` or apostrophes `'` (i.e. `"string"` or `'string'`).

Since these expressions do not affect the structure of the BNF rules, the XML-structure or the diagrams, BNF4OOo does not recognize them as special expressions but treats them as strings. On the other hand `"`, `'`, and `[]` may contain characters used in symbols. Therefore the affected items are parsed literally.

2.1.4 ISO-EBNF

In 1996 the ISO/IEC 14977 was published with the intention to provide a general standard for an Extended Backus-Naur form. It shows that problems occur, if a language described by BNF contains BNF-metasympols. Other approaches designed their own BNF/EBNF in order to describe a certain language which prevents the metalanguage from being used generally. The ISO/IEC approach is based on a suggestion of Niklaus Wirth for extending BNF, and has a detailed documentation¹⁰. [WirthNi]. The following chapters give a brief overview on the ISO-EBNF structures.

2.1.4.1 Defining-symbol

The defining-symbol has the same meaning as in BNF, but instead of `::=` a simple equal character `=` is used.

2.1.4.2 Definition-separator-symbol

BNF4OOo's `XOR` is called definition-separator-symbol by ISO/IEC. The vertical bar `|` is used to separate the alternatives. If `|` cannot be used for any reasons, the characters `/` and `!` can be used instead.

2.1.4.3 Concatenate-symbol

While other BNF-dialects implicitly use the space or tab character to separate the different elements, ISO/IEC explicitly uses a comma character `,` for this purpose. This is only necessary if those separated elements are part of a sequence (the name for a BNF-rule). In BNF 4 the `defined-symbol` can be either `part one` followed by `part two` on the one hand, or `part three` on the other hand. In BNF4OOo the concatenate-symbol is named `delimiter`.

```
defined-symbol = part one, part two | part three
```

BNF 4: ISO/IEC Concatenate-symbol example

¹⁰ C.f. [ISOEBNF01]

2.1.4.4 Terminator-symbol

A sequence does not end with the beginning of the next sequence or the end of line (`eol`) but with the `terminator-symbol`, a semicolon `;`. Alternatively a full stop `.` can be used.

2.1.4.5 Grouped-Sequence

As in XML-EBNF, it is allowed to group elements by enclosing them in simple brackets `(,)`.

2.1.4.6 Optional-Sequence

Optional elements or groups of optional elements are enclosed in square brackets `[,]` or alternatively in `(/` and `/)`.

2.1.4.7 Repeated-sequence

Elements or groups of elements appearing zero or more times are enclosed in curly brackets `{, }` or alternatively in `(:` and `:)`.

2.1.4.8 Syntactic-factor

The syntactic-factor is the counterpart to BNF4OOo's `factor` (see 2.3.17, page 38). If an element has a certain repetition rate, then the rate and an asterisk `*` is written before the element. BNF 5 illustrates the syntactic-factor.

$A = 3*B$

BNF 5: ISO/IEC repetition-symbol example

2.1.4.9 Quote-symbols

In contrast to BNF in ISO-EBNF, not the `Nonterminal` but the `Terminal` elements are marked with quote-symbols by using either apostrophs `'` or quotes `"`.

2.1.4.10 Syntactic-exception-symbol

ISO-EBNF allows the user to list elements which the defined element (the `Nonterminal` element) must not contain. As with XML-EBNF this is done by writing a minus `-` between the element on the left side and the forbidden elements on the right side.

2.1.4.11 Comment-symbols

Comments can be written anywhere in the text by enclosing them between **(*** and ***)**.

2.1.4.12 Special-sequence-symbols

Since ISO/IEC designed their EBNF-dialect for a general purpose, it gives the user the opportunity to define own enhanced structures. This is indicated with the special-sequence-symbol **?**. This option is not implemented in BNF4OOo.

Figure 1 shows all structures with their symbols in a nutshell.

Structure	Normal Symbol	Alternative Symbol
concatenation	,	
definintion	=	
definition-separation		/ or !
grouped-sequence	(and)	
optioal-sequence	[and]	(/ and /)
repeated-sequence	{ and }	(: and :)
syntactic-factor	*	
syntactic-exception	-	
quoted element	' and '	" and "
comment	(* and *)	
special-sequence	?	

Figure 1: Table of ISO/IEC EBNF structures

2.1.5 ABNF

The **A**ugmented **B**NF is one of the newer dialects derived from BNF in 1997. It was written as an Internet standards track protocol to define formal syntaxes in technical specifications. [ABNF01]. Although the main focus of ABNF is to standardize the notation of charac-

ters and character ranges, it also provides new structures. The following sections give a brief overview on the symbols and structures used in ABNF.

2.1.5.1 Terminal Values

In ABNF **Terminals** can be represented by a non-negative integer (e.g. for the ASCII encoding) and an explicit base. The defined bases follow after the **%** character and are binary **b**, decimal **d** and hexadecimal **x**. A group of characters with the same base can be concatenated by using **.**. Alternatively, characters can also be notated as literals enclosed in quotation-marks **"**. Literals, are always case-insensitive in ABNF. [ABNF02], [ABNF03].

Base	Representation in ABNF
Carriage Return (CR) with binary base	%b1101
CR with decimal base	%d13
CR with hexadecimal base	%x0D
2x CR with decimal base	%d13.13
CR as a literal	CR

Figure 2: ABNF character notation example

2.1.5.2 Rule Form

A single equal character **=** is used to separate the name of the rule from the definition. A rule ends with carriage return line feed (**crlf**), no **terminator-symbol** is used. [ABNF02].

2.1.5.3 Concatenation

Values (elements) in an ABNF-rule are concatenated by **linear white space** (space and horizontal tab). [ABNF04].

2.1.5.4 Alternatives

In ABNF, alternatives (called **XOR** in BNF4OOo) are separated by a forward slash character **/**. [ABNF05].

2.1.5.5 Incremental Alternatives

The ABNF notation, allows the user to split complex sequences of alternatives into fragments (called **XOR-definitions** in BNF4OOo). Each of these fragments is written down as a separate rule, starting with a normal defined rule (using **=** as a **definition-symbol**) and followed by rules which use **=/** as a **XOR-definition-symbol**. Hence the three ABNF rules listed in BNF 6, BNF 7 and BNF 8 have the same meaning. [ABNF05].

```
Ruleset = alt1 / alt2 / alt3 / alt4 / alt5
```

BNF 6: ABNF notation for alternatives

```
Ruleset = alt1  
Ruleset =/ alt2  
Ruleset =/ alt3  
Ruleset =/ alt4  
Ruleset =/ alt5
```

BNF 7: ABNF notation for incremental alternatives

```
Ruleset = alt1 / alt2  
Ruleset =/ alt3  
Ruleset =/ alt4 / alt5
```

BNF 8: ABNF notation mix of alternatives and incremental alternatives

2.1.5.6 Value Range Alternatives

A range of alternative numeric values is indicated by using a dash character **-** between the minimum and the maximum value. [ABNF06].

```
DIGIT = %x30-39
```

BNF 9: ABNF value range alternatives example

Since BNF4OOo allows **range-symbols** only to have a circumfix position (see 2.3.11, page 35), this kind of notation is not supported explicitly (e.g. in contrast to the **range-symbols** used in the XML-EBNF with the position **circumfix**). Therefore it is necessary to notate

the whole string (e.g. `%x30-39`) without any `linear white space` to make sure it is parsed as one chunk, otherwise the string will be fragmented by the parser.

2.1.5.7 Sequence Group

Elements can be grouped by enclosing them in paranthesis `(,)`. [ABNF06].

2.1.5.8 Variable Repetition

ABNF uses an asterisk character `*` in front of a string to express the variable repetition of this element. The full form is `<a>*element`. The elements `<a>` and `` are optional and express the minimum and maximum number of repetitions. If `<a>` is left blank, the minimum number is 0. If no `` is not given, there is no upper limit. [ABNF07]

```
sample1 = *element           ;range goes from 0 to infinity
sample2 = 1*4element         ;range goes from 1 to 4
sample3 = *3element          ;range goes from 0 to 3
sample4 = 2*element           ;range goes from 2 to infinity
sample5 = 3*3element          ;range is exactly 3
```

BNF 10: ABNF variable repetitions example

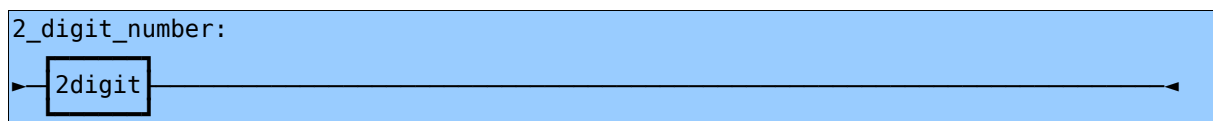
2.1.5.9 Specific Repetition

This is a special case of the variable repetition with a fix repetition rate. It is notated by writing the Repetition rate `<n>` in front of the affected element (e.g. `<n>element`). [ABNF07].

This form of notation is not supported explicitly by BNF4OOo's counterpart, the `factor-instruction` (see 2.3.17, 38). BNF4OOo cannot recognize a range of numbers as a symbol, a new `factor-instruction` would have to be made for every number that could possibly occur. Therefore BNF 11 would be interpreted as shown in Syntax Diagram 1 (SD 1).

```
2_digit_number= 2digit;
```

BNF 11: ABNF specific repetitions example 1



SD 1: ABNF specific repetitions example 1

A workaround for this unclear illustration is to use a variable repetition instead as shown in BNF 12 and SD 2.

```
2_digit_number= 2*2digit;
```

BNF 12: ABNF specific repetitions example 2



SD 2: ABNF specific repetitions example 2

2.1.5.10 Optional Sequence

One or more elements can be marked as optional by enclosing them in square brackets `[`, `]`. [ABNF07].

2.1.5.11 Comment

In ABNF, comments start with a semicolon `;` and end at `EOL`. [ABNF07].

2.1.6 Not Supported by BNF4OOo, xBNF

The "Extreme BNF" is another extension of the BNF which aims to have a notation as brief as possible. [XBNF01]. The following example illustrates the XBNF notation in contrast to BNF and EBNF when dealing with a `loop-instruction`.

Dialect	Rule
BNF	<code><number> ::= <digit> <number><digit></code>
EBNF	<code>number ::= digit {digit}</code>
XBNF	<code>Number ::= N(digit)</code>

Figure 3: Comparison of BNF, EBNF and XBNF syntax

The short and powerful notation of XBNF makes it a handy BNF-dialect. Yet, since to notation differs so much from other BNF-dialects the BNF4OOo parser cannot parse the rules correctly, it would require a modified version¹¹.

2.2 Positions of BNF-symbols

In the literature the different BNF-dialects differ not only by the strings used to describe attributes of elements, but also by the notation of these strings. The sentence "Element A is defined by one or more elements of B" can be written as described in BNF 13 using XML-EBNF.

```
A ::= B+
```

BNF 13: **Suffix** notation example

Using the EBNF-dialect defined in ISO/IEC 14977 the same sentence would be written as shown in BNF 14.

```
A = {B}
```

BNF 14: **Circumfix** notation example

Apart from the different definition strings (**::=** and **=**) the two BNF-dialects use different approaches to describe the repetition of element B. In the first case the **repetition-symbol** consists of one character **+** right after element B while in the second example the symbol consists of two characters, one **{** right before and one **}** right behind the element B.

Since BNF4OOo aims to be as flexible as possible, it separates the BNF-symbols not only from the string used but also from its position. From mathematic and computer science these positional categories for operators are commonly known:

- Prefix (e.g. **+AB**),
- Suffix (e.g. **AB+**),
- Infix (e.g. **A+B**),

¹¹ XBNF might be supported in a following version of BNF4OOo, if there is a demand.

- Circumfix (e.g. `+AB+`).

All of these notations are supported for most of BNF4OOo's `BNF-instructions` (see 2.3, 26). For some instructions not all notations are reasonable. Hence, these instructions are limited to a subset of these notations.

2.3 Supported BNF-Instructions

All BNF-dialects contain structures represented by symbols. In combination with information about the specific position of the used symbols, these structures instruct the reader how to deal with the affected elements. Therefore in BNF4OOo these entities are named `BNF-instructions`. The focus of the following chapters is to describe BNF4OOo's supported `BNF-instructions` in detail.

2.3.1 Definition

The `definition-instruction` can be seen as the root element of every BNF-rule. The element on the left side of the `definition-symbol` is defined by the elements on the right side. BNF 15 uses `::=` as the `definition-symbol`.

```
[1] document ::= prolog element Misc*
```

BNF 15: `Definition-instruction` with comment example

In BNF 15 there are two symbols before the `definition-instruction`. In this case the BNF-Parser assumes that the last element is meant to be defined because it has the closest position to the `definition-symbol`. All elements before are seen as comments and will not be parsed. So the expression of BNF 15 is exactly the same as in BNF 16.

```
document ::= prolog element Misc*
```

BNF 16: `Definition-instruction` without comment example

The definition is the only instruction which is absolutely necessary in BNF4OOo. Its `position` cannot be defined explicitly, it is always between the defined element and the

defining elements. When no `terminator-instruction` is defined in the BNF-dialect, the `defining-symbol` also acts as some kind of `delimiter` between the BNF-rules. The first BNF-rule starts at the beginning of the text and ends at the next found rule containing a `definition-symbol`. If no subsequent BNF-rule is found, it ends at the end of the text. All other BNF-rules start with a rule containing a definition string and end either at the next BNF-rule or at the end of the text.

BNF 17 and 18 make no difference to the parser when no `terminator-instruction` is defined (for examples containing a `terminator-instruction` see 2.3.5, 30).

```
Here is some text
Boolean ::= true
          | false
Number_from_one_to_three ::= 1
                             | 2
                             | 3
```

BNF 17: Splitted BNF-rules example 1

```
Here is some text Boolean ::= true | false
Number_from_one_to_three ::= 1 | 2 | 3
```

BNF 18: Splitted BNF-rules example 2

The `definition-instruction` has two subcategories called `integrated-definition` and `XOR-definition` which are described in the following sections.

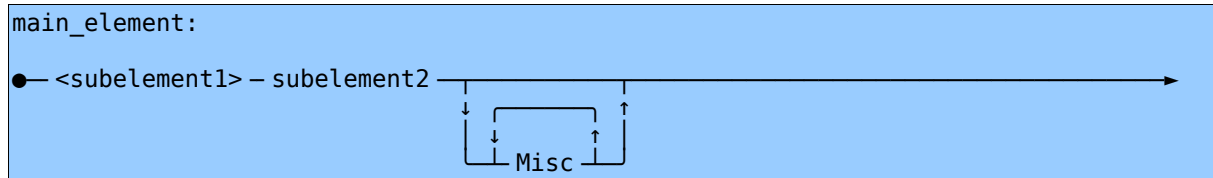
2.3.2 Integrated-Definition

This subtype of the `definition-instruction` is an own customization which may be a handy tool for creating one syntax diagram out of multiple BNF-rules. Using `definition-instructions` creates one syntax diagram for every BNF-rule. If a defined element appears in other BNF-rules as well as part of their definitions, this element gets his own diagram and is printed in these other generated diagrams as well. If this element is marked as a `Nonterminal` by using a `Nonterminal-instruction` then it will be marked in all other diagrams with the used `Nonterminal-symbol`.

BNF 19, SD 3 and SD 4 illustrate this printing logic when using a `definition-instruction`. BNF 19 uses the original BNF extended by a `optional-loop-instruction` using `*` with the position `postfix` (see 2.3.15, 37).

```
main_element ::= <subelement1> subelement2 Misc*
<subelement1> ::= Misc*
```

BNF 19: two related BNF-rules



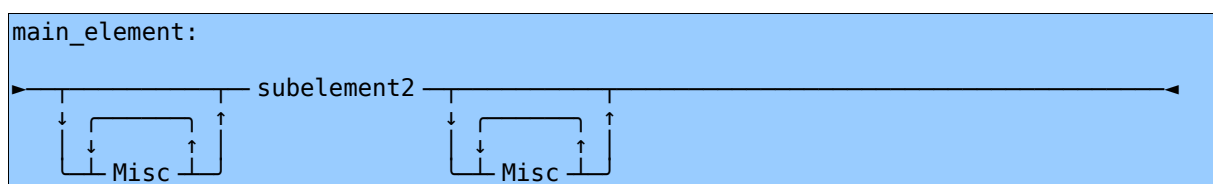
SD 3: First SD created from BNF 19 when using a definition-instruction



SD 4: Second SD created from BNF 19 when using a definition-instruction

In SD 3 `subelement1` is printed with `<` and `>` to express that it is defined in another diagram (in this case SD 4).

If `::=` is defined as an `integrated-definition-symbol` in BNF 19, the exact same BNF-rules create only SD 5. The `subelement1` has vanished completely, its content was integrated in the `main_element` diagram.



SD 5: Integrated-definition example 3

If the element defined by a `integrated-definition-instruction` does not appear in any other BNF-rule, it will be treated as in the `definition-instruction`. If it occurs more than one time in another BNF-rule or in more than one BNF-rule, the element's content will be integrated in every place.

It is advised to use the `integrated-definition-instruction` only for smaller BNFs for two reasons. First it makes complex diagrams hard to read if all BNF-rules are integrated in a single diagram and second due to a bug¹², big BNFs containing many integrated-Definitions may not be printed correctly.

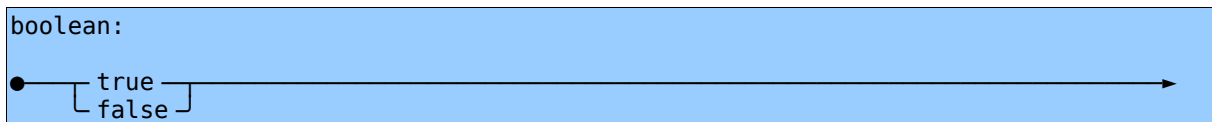
2.3.3 XOR-Definition

The `XOR-definition-instruction` is a mix of a `XOR-instruction` and a `definition-instruction`. In fact it is a `XOR-instruction` spreaded over more than one BNF-rule.

BNF 20 uses `/::=` as a symbol for `XOR-defintions`.

```
boolean ::= true
boolean /::= false
```

BNF 20: `XOR-definition` example 1



SD 6: `XOR-definition` example 1

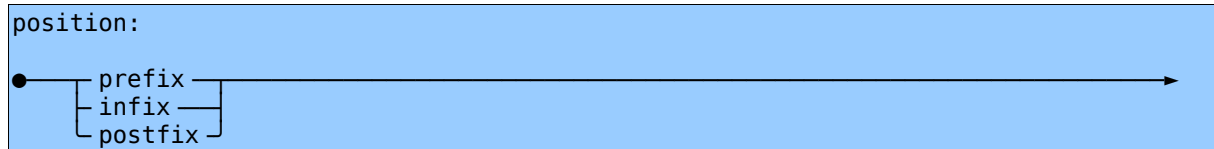
`XOR-defintitions` are always linked to `definition-instructions` or `integrated-definition-instructions`. If no BNF-rule contains such a defined element or if there is more than one defined element, an error will be raised¹³. On the other hand it is valid to link more than one `XOR-definition` to a defined element as BNF 21 and SD 7 show.

```
position ::= prefix
position /::= infix
position /::= postfix
```

BNF 21: `XOR-definition` example 2

¹² When the `integrated-definition-instruction` is used for many BNF-rules, it may occur that the vertical lines are not at the correct position and therefore the whole syntax diagram is unreadable.

¹³ Elements must not be defined in more that one BNF-rule by using `definition-instructions` or `integrated-definition-instructions`.



SD 7: XOR-definition example 2

2.3.4 Delimiter

Some BNF-dialects do not separate elements by space characters but use a **delimiter-instruction**. Cf. [ISOEBNF02]. BNF 22 uses **,** as a **delimiter-symbol**.

```
defined element ::= one component, a second component
```

BNF 22: delimiter-instruction example



SD 8: delimiter-instruction example

If no **delimiter-instruction** is used, then **space** and **tabulator** are used as default **delimiter-symbols**.

2.3.5 Terminator

Some BNF-dialects do use **terminator-symbols** to define the end of a BNF-rule. If a **terminator-symbol** is defined, then all BNF-rules have to use it. Everything standing on the right side of the **terminator-symbol** is seen as a comment and will be erased through parsing. In addition **definition-symbols** do not separate the BNF-rules anymore. The **terminator-symbol** marks the end of the current, and the start of the next, BNF-rule when used.

BNF 23, SD 9 and SD 10 illustrate the functions of a **terminator-instruction**. The used **terminator-symbol** is **;**.

```

element1::=
subelement1 subelement2 subelement3
; this is the end of the first rule
element2
::=
subelement4
subelement5
; this is the end of the second rule

```

BNF 23: Terminator-instruction example

```

element1:
●— subelement1 – subelement2 – subelement3 —————→

```

SD 9: Terminator-instruction example part 1

```

element2:
●— subelement4 – subelement5 —————→

```

SD 10: Terminator-instruction example part 2

2.3.6 XOR

The **XOR-instruction** origins right from the original BNF. [ALGOL]. It contains a group of elements and displays that exactly one element of this group must be chosen.

BNF 24 and SD 11 illustrate the **XOR-instruction** using **|** as the **XOR-symbol** with the position **infix**.

```

boolean::= true | false

```

BNF 24: XOR-instruction example 1

```

boolean:
●— [ true ] —————→
   [ false ]

```

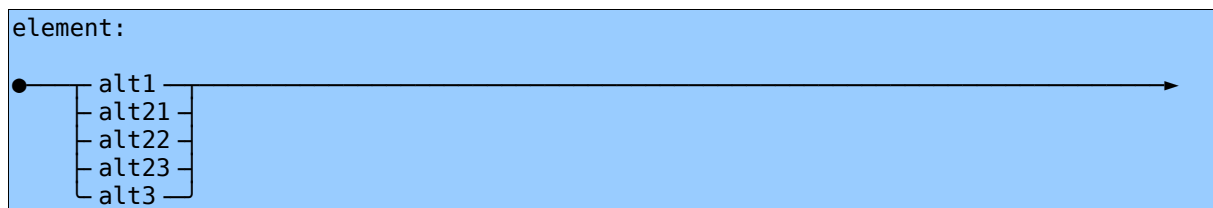
SD 11: XOR-instruction example 1

If a **XOR-instruction** is nested in another **XOR-instruction** i.e. by using a **group-instruction**, BNF400o connects these instructions in the diagram, if possible.

BNF 25 uses **(** and **)** as **group-symbol** with the position **circumfix**.

```
element ::= alt1|(alt21|alt22|alt23)|alt3
```

BNF 25: **XOR-instruction** example 2



SD 12: **XOR-instruction** example 2

If a **XOR-instruction** contains one or more optional elements, the whole **XOR-instruction** becomes optional in the diagram.

BNF 26 uses **[** and **]** as **optional-symbols** with the position **circumfix**.

```
element ::= alt1|[alt2]|alt3
```

BNF 26: **XOR-instruction** example 3



SD 13: **XOR-instruction** example 3

2.3.7 Default

The **default-instruction** is closely related to the **XOR-instruction**. An element with a **default-instruction** inside a **XOR-instruction** means that this element is the default element if no alternative is chosen.

BNF 27 uses `~` as `default-symbol` with the position `circumfix`.

```
element ::= ~alt1~|alt2|alt3
```

BNF 27: `Default-instruction` example



SD 14: `Default-instruction` example

`Default-instructions` should contain data or `Nonterminal` elements only. Otherwise, no error is thrown, but the diagram may not be drawn correctly. This also counts for `XOR-instructions` containing a `default-instruction` which are nested in, or contain an `optional-instruction`. If one of these constellations occur and the diagram is drawn wrongly, it has to be corrected by hand.

2.3.8 Comment

BNF4OOo allows for two different categories of comments. On the one hand they can be implicit, i.e. all elements behind a `terminator-symbol` are seen as comments. On the other hand they can be produced by `comment-instructions`. This chapter covers the second case.

Nearly all common known BNF-dialects use `comment-instructions`. BNF4OOo allows comments to be in a prefix or circumfix position. When using the prefix position, a comment starts with the `comment-symbol` and ends at `EOL` (not the BNF-rule). With the circumfix position, comments start at the `start-symbol` and end at the `end-symbol` no matter if they are actually on the same line or not.

BNF 28, SD 15 and SD 16 show the functionality of `comment-instructions` using `#` as a `comment-symbol` with the position `prefix` and `/*` and `*/` with the position `circumfix`.

```

element  ::= sub1 sub2 # here is a comment
           sub3
element2 ::= sub4 /*
here is another comment text
*/
sub5

```

BNF 28: `Comment-instruction` example

```

element:

```

```

●— sub1 – sub2 – sub3 —————→

```

SD 15: `Comment-instruction` example part 1

```

element2:

```

```

●— sub4 – sub5 —————→

```

SD 16: `Comment-instruction` example part 2

2.3.9 Nonterminal

Some BNF-dialects distinguish between `Terminal` and `Nonterminal` elements. Cf. [ALGOL]. This difference can be made in BNF4OOo as well if preferred, but is not compulsory. Since BNF4OOo knows that `Nonterminal` elements can only contain data, everything inside the instruction's range is treated as such. Also space characters are allowed even if no other `delimiter-instruction` is defined. Therefore `nonTerminal-instructions` may be used to mark literal content as well. The only position allowed for `Nonterminal-instructions` is `circumfix`.

BNF 29 and SD 17 illustrate the functionality of `Nonterminal-instructions` using the symbol `!`.

```

'a BNF-rule' ::= 'a defined element' '::~=' 'the definition'

```

BNF 29: `Nonterminal-instruction` example

a BNF-rule:

● 'a defined element' - '::~' - 'the definition' —————→

SD 17: **Nonterminal-instruction** example

2.3.10 Literal

All strings affected by a **literal-instruction** are parsed as one element. **Literal-instructions** can only have the position **circumfix**.

2.3.11 Range

The EBNF-XML-dialect and ABNF use **range-instructions** to define valid characters for certain elements. In BNF4OOo the functionality of **range-instructions** is closely related to the **literal-instruction**. BNF 30 uses **[** and **]** as **range-symbols**.

```
letter ::= [a-zA-Z]
```

BNF 30: **Range-instruction** example

letter:

● [a-zA-Z] —————→

SD 18: **Range-instruction** example

The position is limited to **circumfix** (e.g. **a-z** would not be possible).

2.3.12 Group

Group-instructions exist in nearly all BNF-dialects. Since they are commonly used with the circumfix position only, BNF4OOo is limited to this position as well.

BNF 31 uses **(** and **)** as **group-symbols** and **+** as a **loop-symbol** with the position **postfix**.

```
element ::= (sub1 sub2)+ ((sub3 sub4)|sub5)
```

BNF 31: **Group-instruction** example



SD 19: Group-instruction example

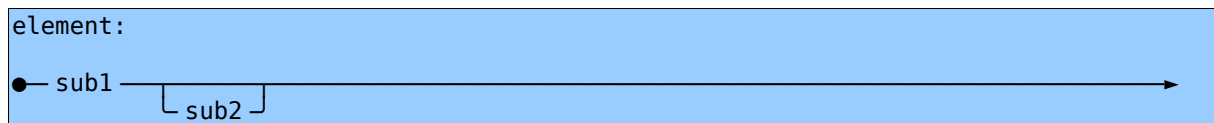
2.3.13 Optional

Optional elements are widely used in nearly all BNF-dialects.

BNF 32 uses `?` as an optional-symbol with the position postfix.

```
element ::= sub1 sub2?
```

BNF 32: Optional-instruction example



SD 20: Optional-instruction example

2.3.14 Loop

A loop-instruction specifies that the affected element(s) can appear one or more time. It does not give any information about the number of iterations.

BNF 33 uses `+` as an optional-symbol with the position postfix.

```
element ::= sub1+
```

BNF 33: Loop-instruction example



SD 21: Loop-instruction example

2.3.15 Optional-loop

The `optional-loop-instruction` combines the functionality of an `optional-instruction` with the functionality of a `loop-instruction`. The whole loop is optional.

BNF 34 uses `*` as an `optional-symbol` with the position `postfix`.

```
element ::= sub1*
```

BNF 34: `Optional-loop-instruction` example



SD 22: `Optional-loop-instruction` example

2.3.16 Argument

The `argument-instruction` functionality is an own custom extension to the `loop-instruction` and the `optional-loop-instruction`. In BNF4OOo `argument-instructions` can help to reduce overhead and therefore improve readability in complex loops. Often times loops contain a certain character (e.g. `,`) at the start of every repetition. BNF4OOo allows the user to define this character as an argument for the loop and moves it away from the loop's baseline up to the loop-line.

BNF 35 and SD 23 show a `loop-instruction` with BNF 35 using `{` and `}` as symbols without an `argument-instruction`. The loop contains a `XOR-instruction` using `|`.

```
element ::= {, string |, int | , long}
```

BNF 35: `Argument-instruction` example 1

SD 23: **Argument-instruction** example 1

The character **,** is repeated in every option. With an **argument-instruction** using the symbol **\$** with the position **prefix** the BNF-rule and the generated diagram look as shown in BNF 36 and SD 24.

```
argument ::= {$( string | int | long )}
```

BNF 36: **Argument-instruction** example 2SD 24: **Argument-instruction** example 2

2.3.17 Factor

This subtype of the **loop-instruction** enables loops with a fix repetition rate. In the diagram, the factor is written in the upper line of the loop. The only position allowed for the **factor-instruction** is **infix**.

BNF 37 uses ***** as the symbol for the **factor-instruction**.

```
element ::= 3*(sub1 | sub2 | sub3)
```

BNF 37: **Factor-instruction** example



SD 25: Factor-instruction example

2.3.18 Complex-factor

The `complex-factor-instruction` is a special case of the `factor-instruction`. It enables the user not only to specify a certain repetition rate, but also to express a range for the repetition rate by setting a lower and an upper limit. If no lower limit is given, it is assumed to be 0, the content of the loop is optional. If no upper limit is given, the repetition rate is unlimited.

BNF 38 uses `*` as a `complex-factor-symbol`.

```
element ::= 1*3(sub1 | sub2 | sub3)
```

BNF 38: Complex-factor-instruction example



SD 26: Complex-factor-instruction example

2.3.19 Exception

Some BNF-dialects such as the ISO EBNF and the XML EBNF dialect use `exception-instructions` to exclude elements from a defining rule. [ISOEBNF02], [W3CXML]. The term exception used in BNF4OOo derives from the ISO EBNF term "syntactic-exception".

BNF 39 uses `-` as the `exception-symbol`.

```
any_cipher_but_not_zero ::= any_cipher - 0
```

BNF 39: **Exception-instruction** example

any_cipher_but_not_zero:

any_cipher - 0

SD 27: **Exception-instruction** example

2.3.20 Overview on the Supported BNF-Instructions

Figure 4 gives an overview over the different BNF-dialects and DITA (see 2.5, page 46) with their explicitly supported **BNF-instructions**.

BNF-instruction	BNF	XML-EBNF	ISO-EBNF	ABNF	DITA
Definition	✓	✓	✓	✓	
Integrated-definition					
XOR-Definition			✓		
Delimiter			✓		
Terminator			✓		
XOR	✓	✓	✓	✓	✓
Default					✓
Comment		✓	✓	✓	✓
Nonterminal	✓			✓	✓
Range		✓		✓	
Group		✓	✓	✓	✓
Optional		✓	✓	✓	✓
Loop		✓	✓	✓	✓
Optional-Loop		✓	✓	✓	✓
Argument					✓
Factor			✓	✓	
Complex-factor				✓	
Exception		✓	✓		

Figure 4: Overview of BNF-dialects and DITA

2.4 The Syntax Diagram Types

Since the introduction of the Backus-Naur-Form syntax diagrams represent a graphical alternative to BNF. Early books using syntax diagrams include the "Pascal User Manual"

BNF 40 is based on the XML-EBNF standard, extended with a `default-instruction`, using the symbol `$` with the position `prefix` and a `Nonterminal-instruction` with `<` and `>`.

```
/*$ is used as a default-symbol with the position prefix*/
Commands::=(OPEN ($ (BOTH <Write Options>)|READ|(WRITE <Write Options>))(<Options>?))
|CLOSE
|FLUSH
|(SEEK|POSITION)($=|'<'|'+';) offset (READ|WRITE)?($CHAR|LINE)
|QUERY ( DATETIME
|EXISTS
|HANDLE
|(SEEK|POSITION) ( (READ ($CHAR|LINE))
| (WRITE ($CHAR|LINE))
| SYS
)
|SIZE
|STREAMTYPE
|TIMESTAMP
)

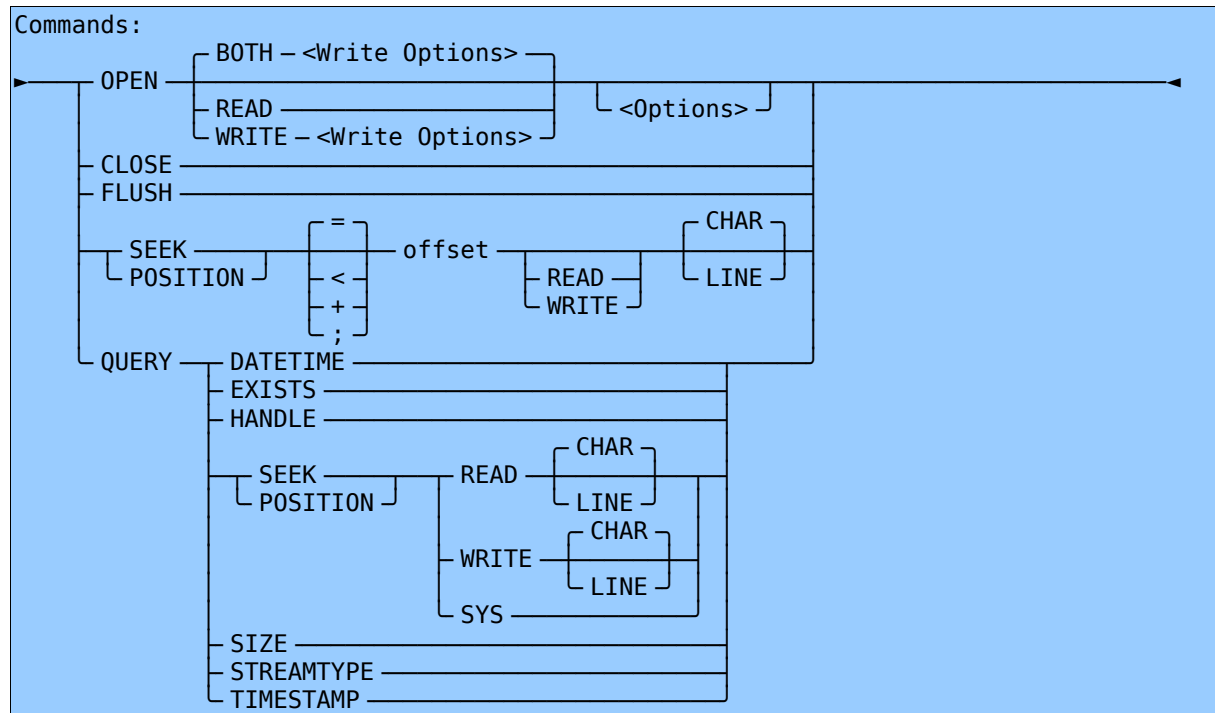
<Write Options> ::= (APPEND|REPLACE)?

<Options> ::= (SHARED|SHAREREAD|SHAREWRITE) (NOBUFFER|BINARY (RELENGTH length)?)+
```

BNF 40: Command strings in streaming functions

The next chapters show the different syntax diagram types that can be created by BNF4OOo. All diagrams are drawn using characters. The main advantages of this approach is that it provides a maximum of flexibility. First of all every diagram can be extended or changed manually by typing. This way for example comments can be written without much effort. In addition every character used in the diagram can be changed in BNF4OOo's Graphical User Interface (GUI) `option-GUI` (see 2.7.4, page 62). Character based diagrams can be used in any media type from simple `.txt` files over system shells up to presentation files and can be scaled without any quality loss. If Unicode characters are not supported, ASCII characters can be used. This way the diagrams can be inserted directly in the code as a comment for example. When an image format is preferred, the character-based diagram can be exported using the `ExportDiagram` function (see 2.6.6, page 54).

The main disadvantage of character based diagrams is that only monospace based fonts can be used, otherwise the drawn diagram is not readable. When using Unicode characters, only a few font types support the characters for drawing lines. BNF4OOo allows the user to select any font available in OOo (see 2.7.2, page 58) but only monotype based fonts are qualified. Another problem which appeared during the programming process is that Unicode diagrams are not displayed correctly all the time. For example in OOo, de-



SD 29: Command strings in streaming functions – Unicode image

2.4.3 Boxed Unicode Image

The boxed Unicode image is the most complex diagram type in BNF4OOo. All items are framed which makes this diagram type easy to read. On the other hand this type of diagram needs three times the space of the other ones. The font size used in SD 30 was reduced to 6 manually to make it fit into one page.

compared to other approaches. Therefore, the BNF-rules can be transferred not only into diagrams, but also into a XML format for further processing. On the other side, BNF4OOo can be used to transfer the BNF4OOo XML format into a BNF-dialect, if the desired BNF-dialect contains all used **BNF-instructions**.

The structure of the XML format derived from IBM's DITA (**D**arwin **I**nformation **T**yping **A**rchitecture) which is an XML based end-to-end architecture for authoring, producing, and delivering technical information. [DITA01]. The DITA programming domain has a broad set of tags for creating syntaxdiagrams. BNF4OOo uses a subset of these tags so the structure looks quite familiar. But since not all off **BNF-instructions** are supported by DITA (see figure 4, page 39) DITA has been extended at some points.

BNF 41 shows a simple BNF-rule and XML 1 the BNF4OOo XML counterpart. In this chapter the XML-EBNF dialect will be used for all examples.

```
Element ::= sub1 sub2 sub3
```

BNF 41: Illustration of the BNF4OOo XML format

```
<?XML version="1.0" encoding="UTF-8"?>
  <!-- Original BNF rules:
    Element ::= sub1 sub2 sub3*
  -->
<syntaxdiagram XMLns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noName-
spaceSchemaLocation="/home/franz/workspace/BNF400o/XML/BNF400oXML.xsd">
  <title>Element</title>
  <groupseq>
    <var>sub1</var>
    <var>sub2</var>
    <var>sub3</var>
  </groupseq>
</syntaxdiagram>
```

XML 1: Illustration of the BNF4OOo XML format

The head of the XML documents consists of two parts, the compulsive first **<?XML ?>** tag with the version and the encoding info and second a comment tag **<!-- -->** containing the BNF-rules of the original format. The reason for keeping the original rules inside the new format is to prevent additional information inside the BNF, such as comments, from getting deleted. This way they are still readable for the human eye.

The main container of the syntax diagram is the `<syntaxdiagram>` tag containing information on the XML Schema with the fully qualified path. [DITADoc01]. The BNF content starts with the `<title>` tag which contains the first defined element. In DITA this is an optional element, but in BNF400o it is mandatory. [DITADoc02]. All following syntax elements refer to this title element. The title tag's scope ends with the next `<title>` tag, a `<fragment>` tag (will be explained later on) or with `</syntaxdiagram>`.

BNF 42 contains two unrelated BNF rules. Both are merged in one tree in XML 2.

```
element1 ::= sub1 sub2
element2 ::= sub3 sub4
```

BNF 42: Two unrelated BNF rules in one XML file

```
<?XML version="1.0" encoding="UTF-8"?>
  <!-- Original BNF rules:
    element1 ::= sub1 sub2
    element2 ::= sub3 sub4
  -->
  <syntaxdiagram xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noName-
    spaceSchemaLocation="/home/franz/workspace/BNF400o/XML/BNF400oXML.xsd">
    <title>element1</title>
    <groupseq>
      <var>sub1</var>
      <var>sub2</var>
    </groupseq>
    <title>element2</title>
    <groupseq>
      <var>sub3</var>
      <var>sub4</var>
    </groupseq>
  </syntaxdiagram>
```

XML 2: Two unrelated BNF rules in one XML file

All sequences of elements describing the title element are nested inside `<groupseq>` tags. With the attribute `importance="optional"`, the whole content of the sequence is optional. [DITADoc03].

DITA's equivalent to the `XOR-instruction` is the `<groupchoice>` tag. All elements nested inside are alternatives. Groupchoices can also have the attribute `importance="optional"`. [DITADoc04]. If a grouchoice contains a default value, it is marked with the attribute `importance="default"`. [DITADoc05].

XML 3 contains all the elements described by now.


```
/*This XML-EBNF dialect had been enhanced, the $ char mark the default value in
a xor */
element ::= sub1 ($sub2 | sub3) sub4?
```

BNF 43: Example for `groupchoices` and `optional sequences`

```
<?XML version="1.0" encoding="UTF-8"?>
<!-- Original BNF rules:
/*This XML-EBNF dialect had been enhanced, the $ chars mark the default value in
a xor */
element ::= sub1 ( $sub2$ | sub3) sub4?
-->
<syntaxdiagram XMLns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noName-
spaceSchemaLocation="/home/franz/workspace/BNF400o/XML/BNF400oXML.xsd">
<title>element</title>
<groupseq>
<var>sub1</var>
<groupchoice>
<var importance="default">sub2</var>
<var>sub3</var>
</groupchoice>
<groupseq importance="optional">
<var>sub4</var>
</groupseq>
</groupseq>
</syntaxdiagram>
```

XML 3: Example for `groupchoices` and `optional sequences`

If the first tag inside a `<groupseq>` or a `<groupchoice>` is a `<repseq>` tag, the whole sequence can be repeated. In DITA the `repseq` tag can either be empty `<repseq/>` or can contain a separator character which must be used between the repetition of the syntax elements. [DITADoc06]. The DITA syntax diagram approach knows only unspecified repetitions meaning that no repetition rate is given. In BNF400o on the other hand, if a repetition rate is needed, it can be expressed by writing `<repseq><kwd> "repetition rate" </kwd><repseq>`. The term `"repetition rate"` can be a single number (i.e. 3) or a range using the notation of the complex-factor (i.e. 1*3, 1*, *3).

XML 4 contains an unspecified loop and another one with a factor.

```
/*This XML-EBNF dialect had been enhanced, 3*
means that the element must repeat three times */

element ::= sub1+ 3*(sub2)
```

BNF 44: Example for loops in DITA

```
<?XML version="1.0" encoding="UTF-8"?>
  <!-- Original BNF rules:
  /*This XML-EBNF dialect had been enhanced, 3*
  means that the element must repeat three times */

  element ::= sub1+ 3*(sub2)
  -->
<syntaxdiagram xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noName-
spaceSchemaLocation="/home/franz/workspace/BNF400o/XML/BNF400oXML.xsd">
  <title>element</title>
  <groupseq>
    <groupseq>
      <repsep/>
      <var>sub1</var>
    </groupseq>
    <groupseq>
      <repsep> <kwd>3*</kwd></repsep>
      <var>sub2</var>
    </groupseq>
  </groupseq>
</syntaxdiagram>
```

XML 4: Example for loops in DITA

The `exception-instruction` is not supported by DITA. Therefore BNF4OOo enhances `<groupseq>` by the attribute `noElement`. The value of this attribute contains all BNF elements that are excepted from the sequence as illustrated in XML 5.

```
element ::= sub1 - sub2
```

BNF 45: Example for exceptions – not supported by DITA

```
<?XML version="1.0" encoding="UTF-8"?>
  <!-- Original BNF rules:
  element ::= sub1 - sub2
  -->
<syntaxdiagram xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noName-
spaceSchemaLocation="/home/franz/workspace/BNF400o/XML/BNF400oXML.xsd">
  <title>element</title>
  <groupseq>
    <groupseq noElement="sub2">
      <var>sub1</var>
    </groupseq>
  </groupseq>
</syntaxdiagram>
```

XML 5: Example for exceptions – not supported by DITA

If the defined element of a BNF-rule appears in one or more other BNF-rules, it is stored in a `<fragment>` container. Every time this element appears, it is linked to this fragment by

using `<fragref href="#ID"/>` as illustrated in BNF 46, XML 6, SD 31, SD 32 and SD 33. [DITADoc06].

```
<element1> ::= sub1 <sub2>
<element2> ::= sub3 <sub2>
<sub2> ::= misc
```

BNF 46: Example for fragments

```
<?XML version="1.0" encoding="UTF-8"?>
  <!-- Original BNF rules:
  element1 ::= sub1 sub2
  element2 ::= sub3 sub2
  sub2 ::= misc
  -->
<syntaxdiagram xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noName-
spaceSchemaLocation="/home/franz/workspace/BNF400o/XML/BNF400oXML.xsd">
  <title>element1</title>
  <groupseq>
    <var>sub1</var>
    <fragref href="#1"/>
  </groupseq>
  <title>element2</title>
  <groupseq>
    <var>sub3</var>
    <fragref href="#1"/>
  </groupseq>
  <fragment id="#1">
    <title>sub2</title>
    <groupseq>
      <var>misc</var>
    </groupseq>
  </fragment>
</syntaxdiagram>
```

XML 6: Example for fragments

```
element1:
```

```
► sub1 – <sub2> ◀
```

SD 31: Example for fragments part 1

```
element2:
```

```
► sub3 – <sub2> ◀
```

SD 32: Example for fragments part 2

2.6.3 XML2BNF

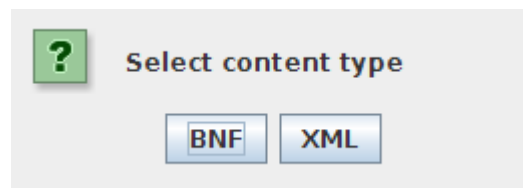


Figure 9: Import data content dialog window

2.6.4 BNF2AII

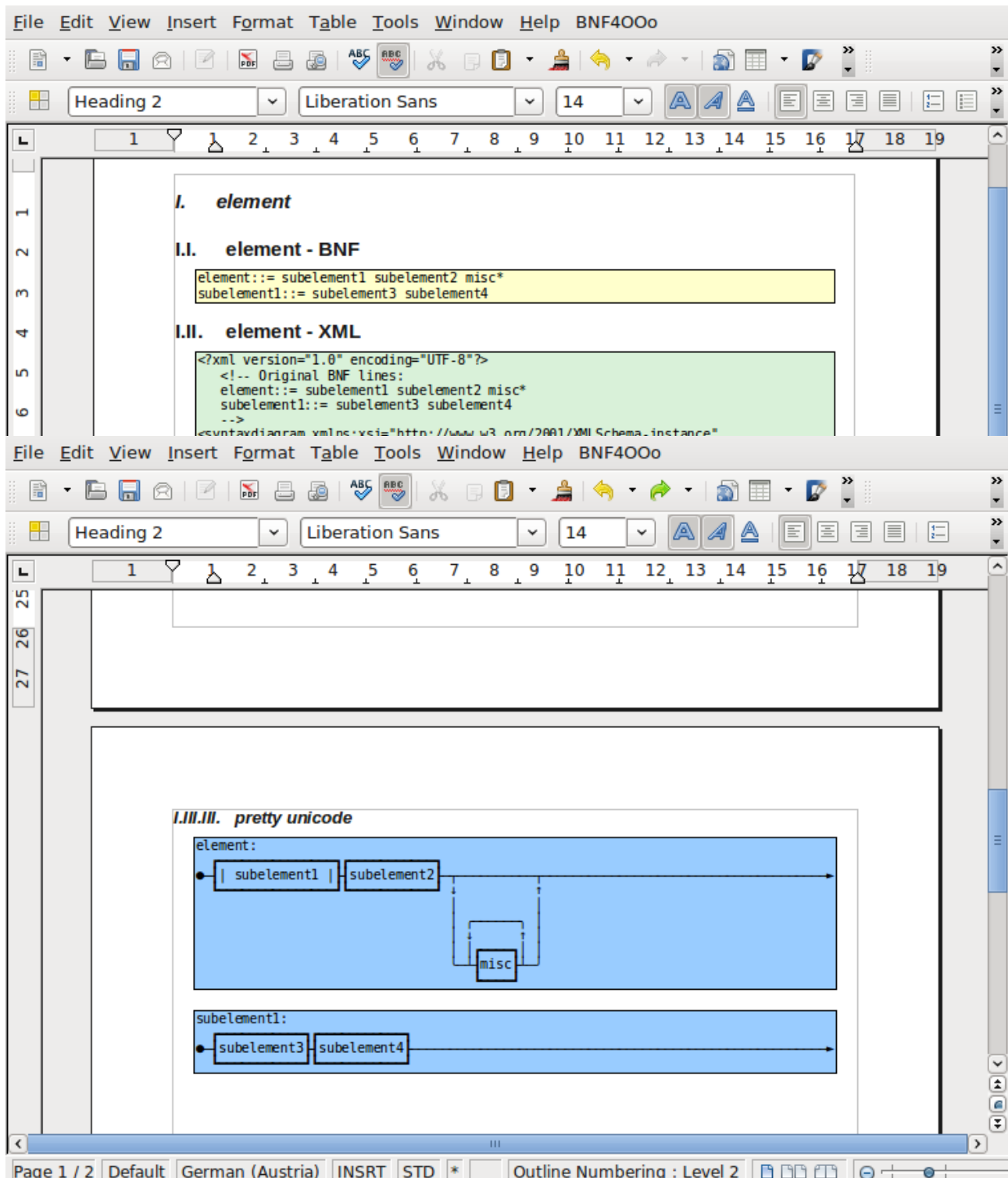
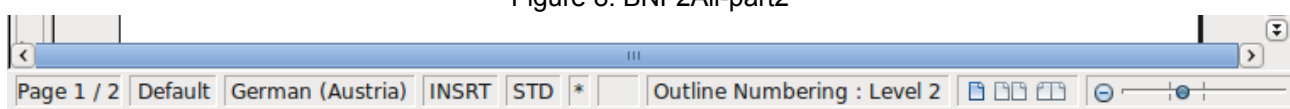


Figure 8: BNF2All-part2



In the following window one or more files can be selected. Every file type can be chosen but the default types are `*.bnf`, `*.ebnf`, `*.abnf` and `*.txt` for BNF, `*.XML` and `*.txt` for XML.

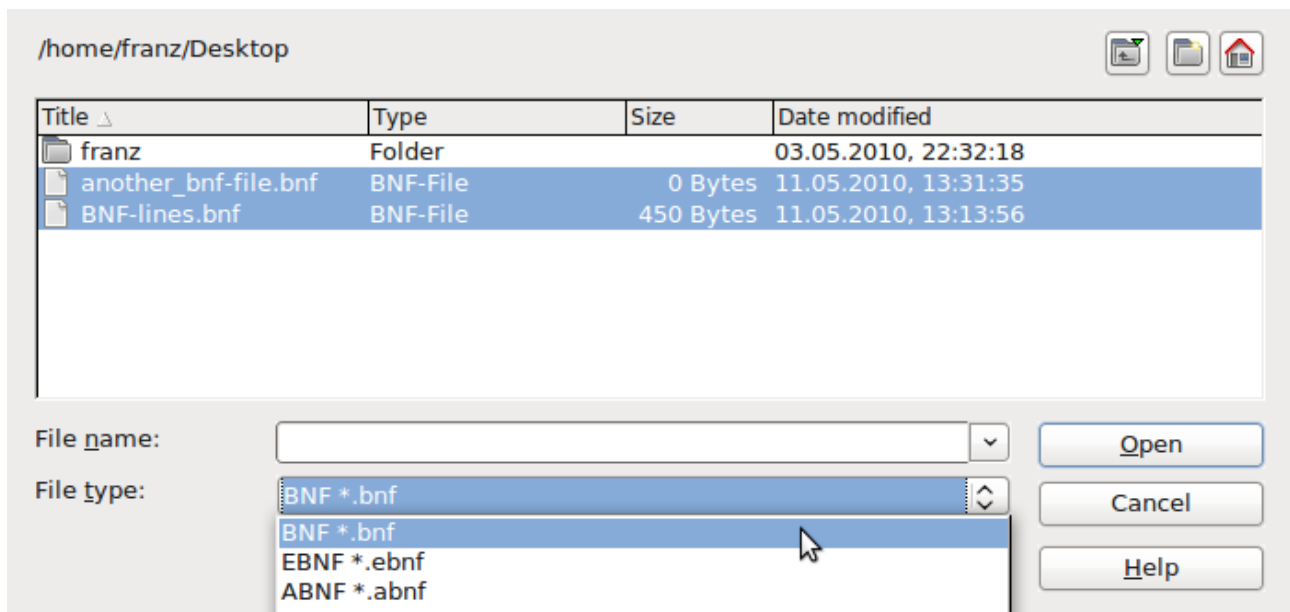


Figure 10: Import data dialog window

Now the content of the selected files is imported at the actual cursor position. If selected, additional information (file name, time stamp and the full path) are written directly above the newly created paragraphs. This setting can be chosen in the `General Options` (see 2.7.1, page 56).

2.6.6 Export Diagram

Clicking the `export diagram` button starts the script `OOoExportDiagram.rex`. With this script all paragraphs with the style "diagram" are exported as images by using `sDraw` (the OOo drawing program). Since not the whole paragraph but only the text is copied and modified in `sDraw`, the font and background colors are not exported but set to black and white.

2.6.7 Options

The `options button` starts the script `OptionGUI.rex` which allows the user to change all settings in BNF4OOo in a comfortable way (see 2.7, page 55).

2.7 BNF4OOo OptionGUI

The BNF4OOo option graphical user interface can be started either inside the BNF4OOo toolbar inside a `sWriter` document (see 2.6, page 52) or directly by running the script `OptionGUI.rex` in a Linux/Mac shell or a Windows Command file (`cmd/shell`). The script is written in ooRexx but uses Java classes by using the BSF4ooRexx class `BSF.CLS`.

The GUI works as an interface between the user and the different property files to make all settings in BNF4OOo editable in a comfortable way. If wanted, the property files can be changed directly, but since the properties are not documented in detail it should be avoided (a quick overview on the BNF4OOo files can be found in 6.3, page 94). Figure 11 gives an overview about the affected property files and their purpose.

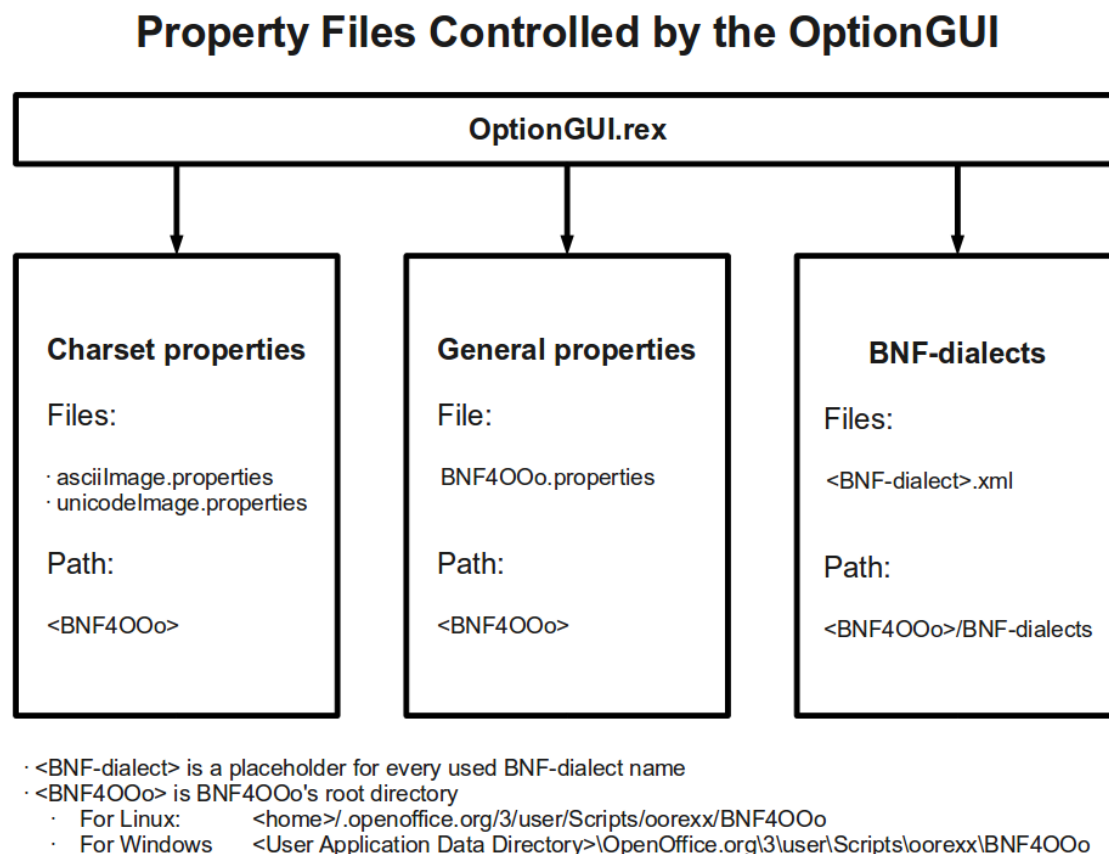


Figure 11: OptionGUI properties

When starting the GUI a window is built with two main elements, the content frame on the right side which shows the different editable options and the menu bar on the left. A click

on the button shows the related options in the content frame. As shown in figure 12 buttons are named **General Options**, **Appearance**, **BNF-dialect**, **Drawing-symbols** and **Export Options**.

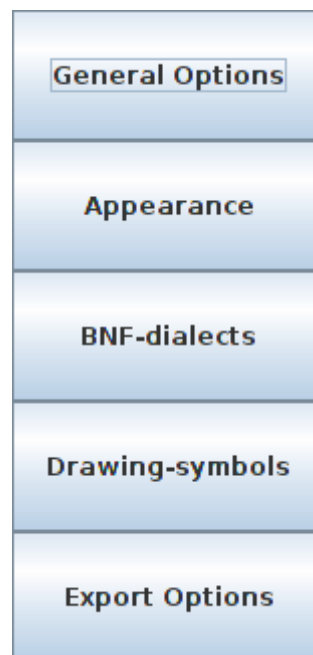


Figure 12: **OptionGUI** menu bar

The related options are explained in detail in the next chapters. At the start the general options are displayed but all other content frames are loaded as well. Hence, it may take some seconds until the window is loaded and displayed depending on the number of BNF-dialects used. But when loaded the GUI reacts quickly to user input. The advantage of loading all options at the start is that the user can switch the activated content panels without losing unsaved changes in the former activated panel.

Each content frame contains a **Save** and a **Quit** button at the bottom line. Clicking on the **Save** button, saves the current settings in this content frame only, all other content frame's settings are untouched. In contrast to this the **Quit** button closes the whole window.

Most options only make sense in the **sWriter** context but some also have an effect, if BNF4OOo is executed in a **cmd/shell**. If this is the case, it is mentioned in the chapter.

2.7.1 General Options

The **General Options** affect the behavior of the BNF4OOo scripts essentially. Figure 13 shows the activated content frame.



The image shows a 'General Options' dialog box with a light gray background and a blue border. At the top, the title 'General Options' is in a blue header bar. Below the title, there are five settings:

- Select BNF-dialect:** A dropdown menu with 'EBNF-XML' selected.
- Choose range for actions:** Three radio buttons: 'document' (selected), 'paragraph', and 'selection'.
- Choose diagram type:** Three radio buttons: 'ASCII', 'Unicode' (selected), and 'boxed-Unicode'.
- Use all diagram types in BNF2All**: A checkbox labeled 'yes' which is checked.
- Print file information when importing data**: A checkbox labeled 'yes' which is checked.

At the bottom of the dialog, there are two buttons: 'Save' on the left and 'Quit' on the right, both with blue backgrounds and white text.

Figure 13: General Options

Select BNF-dialect

The selected BNF-dialect is used for all BNF4OOo functions even when running in a `cmd/shell`. All .XML files stored in the `BNF-dialects` folder are loaded into the `JComboBox`.

Choose range for actions

This selection allows to determine where BNF4OOo looks for input. If `document` is selected, all paragraphs of the paragraph type (e.g. `BNF`) are transformed to the output paragraph, (e.g. `diagram`). On the other hand the option `paragraph` only transforms the first matching paragraph found in the document (not the first paragraph

below the current cursor position!). This is useful if the user wants to transfer the document step by step. Finally `selection` uses the current selection at the current cursor position as input, no matter what paragraph style this selection has.

Choose diagram type

Here the used diagram type `ASCII`, `Unicode` and `boxed-Unicode` can be chosen (see 2.4, page 41).

Use all diagram types in BNF2All

If selected the script `OooBNF2All.rex` will render the BNF input into all available diagram types (`ASCII`, `Unicode` and `boxed-Unicode`), one after another. If not selected only the activated diagram type is used (e.g. `Unicode`).

Print file information when importing data

The script `OooImportData.rex` will print file information above the imported paragraph in `sWriter`, if this option is selected (see 2.6.5, page 53).

2.7.2 Appearance Options

The `Appearance Options` contain the sections `Paragraph Style`, `Diagram Options` and `Numbering Options` which affect the look of the input and output paragraphs in BNF4OOo. Figure 14 shows the appearance option's content frame.

Appearance Options	
Paragraph Styles	
BNF Font: <input type="text" value="DejaVu Sans Mono"/> Font Size: <input type="text" value="9"/> Font Color: <input type="text" value="000000"/> Background Colo... <input type="text" value="FFFFCC"/>	diagram Font: <input type="text" value="DejaVu Sans Mono"/> Font Size: <input type="text" value="9"/> Font Color: <input type="text" value="000000"/> Background Colo... <input type="text" value="99CCFF"/>
XML Font: <input type="text" value="DejaVu Sans Mono"/> Font Size: <input type="text" value="9"/> Font Color: <input type="text" value="000000"/> Background Colo... <input type="text" value="D9F1D9"/>	error Font: <input type="text" value="DejaVu Sans Mono"/> Font Size: <input type="text" value="9"/> Font Color: <input type="text" value="000000"/> Background Colo... <input type="text" value="FF9696"/>
Diagram Options	
Minimum length of diagram:	<input type="text" value="81"/>
Numbering Options	
Numbering style for 1. level	<input type="text" value="ARABIC 4"/>
Numbering style for 2.level	<input type="text" value="ARABIC 4"/>
Numbering style for 3.level	<input type="text" value="ARABIC 4"/>
<div>Save</div> <div>Quit</div>	

Figure 14: Appearance Options

Paragraph Styles

This section was created to make all paragraph styles used in BNF4OOo editable as easy as possible even if the user is not familiar with OpenOffice. All options can be set in the `sWriter` paragraph options as well. For each paragraph style the `Font`, `Font Size`, `Font Color` and `Background Color` can be changed. All changes are shown in the preview box on the right side. Even if allowed it is not advised to change the font of the paragraph style `diagram`. Firstly, it is absolutely necessary to use a monospace font, otherwise the diagram will not be readable. Secondly, not all font types support the character sets can be used to print the diagrams when using `Unicode image` or `boxed-Unicode image`.

Diagram Options

The option section allows the user to change the `Minimum Length of Diagram` measured in characters. If a diagram's base line is shorter than the setted value, it gets expanded automatically. SD 34 and SD 35 illustrate this option.



SD 34: Diagram with a minimum length set to 0



SD 35: Diagram with minimum length set to 81

If no expansion is wanted, the value should be set to 0. The minimum length option is not only used in `sWriter` but also when BNF4OOo is running in a `cmd/shell`.

Numbering Options

The script `OOoBnf2All.rex` uses headings with the numbering option activated to structure the generated content. The numbering options allow the user to specify the styles of the different numbering levels.

2.7.3 BNF-dialects

This is the most complex frame in the BNF4OOo options. The user can create, edit and remove the BNF-dialects available to BNF4OOo. Although these changes can be made in the XML-files in the subdirectory `BNF-dialects` directly, it is advised to use this interface. The advantage is that all changes to the XML files are done in a controlled way. All generated files are valid whereas any direct change in the document directly can produce an invalid document. For example there are certain instructions in BNF4OOo which only allow a restricted range of positions (see 2.3, page 26).

The BNF-dialect's content frame is divided into `Name`, `Definitions`, `Structural Instructions` and `Instructions`. Figure 15 shows the activated content frame.

BNF-ALGOL_60 customBNF-FRH EBNF-ISO-IEC_14977 EBNF-XML new Set

Name: BNF-ALGOL_60 remove this dialect

Definitions

Type: definition Symbol: :: = delete definition

Add definition

Structural Instructions

Delimiter: (If Field is empty then space/tab is used)

Terminator: (If Field is empty then BNF-rule ends before next defined line or EOF)

Instructions

Type: xor Position: infix Symbol: | delete

Type: nonTerminal Position: circumfix Symbol: < Symbol: > delete

Add instruction

Save Quit

Figure 15: BNF-Dialects

Name

The textfield right to the label **Name:** determines the actual name of this BNF-dialect. If the name of an existing dialect changes, the old dialect will not be replaced but a new set with the new name is added to the collection. All dialects are saved in the subdirectory **BNF-dialects** with their listed name and the **.XML** suffix. The button **remove this dialect** deletes the active dialect from the menu and the hard disk.

Definitions

Definitions play a special role in BNF4OOo. Therefore they have their own section in the `BNF-dialects` GUI. Every `BNF-dialect` needs at least one `definition-instruction` or `integrated-definition-instruction`. BNF4OOo does not set an upper limit for the number of used `definition-instructions`, but it is advice to use as few as possible to keep the BNF-rules readable.

Structural Instructions

This sections contains two instruction types which are essential for parsing the BNF-rules. If the `delimiter`'s text field is left empty, all elements will be delimited by space and tab characters (see 2.3.4, page 30). If no terminator string is given then the `definition-instructions` act as "rule delimiter" (see 2.3.5, page 30).

Instructions

All other instructions available in BNF4OOo are settable in this section. First the instruction's `Type` should be chosen from the dropdown box on the left. Depending on the chosen instruction, the `Position` dropdown box shows a range of positions to choose from. Finally the `Symbol` for the instruction can be entered in the textfields. Most instructions have only one symbol but when the position circumfix is selected, the instruction needs a starting symbol and an ending symbol.

2.7.4 Drawing-symbols

One of BNF4OOo's goals is to be as flexible as possible. Therefore BNF4OOo allows the user to change any character used in the different diagram types. In the `Drawing-symbols` frame each image type has its own tab. Since the Unicode and the boxed Unicode images use the same set of symbols, there is only one tab for Unicode images. Inside the tab frame there is a line for every character showing the name on the left side, a textfield for entering the value on the right side and a preview in the middle. The preview changes as soon as a new value is entered and the textfield is losing focus. Figure 16 shows the image character set's content frame with the Unicode tab activated.

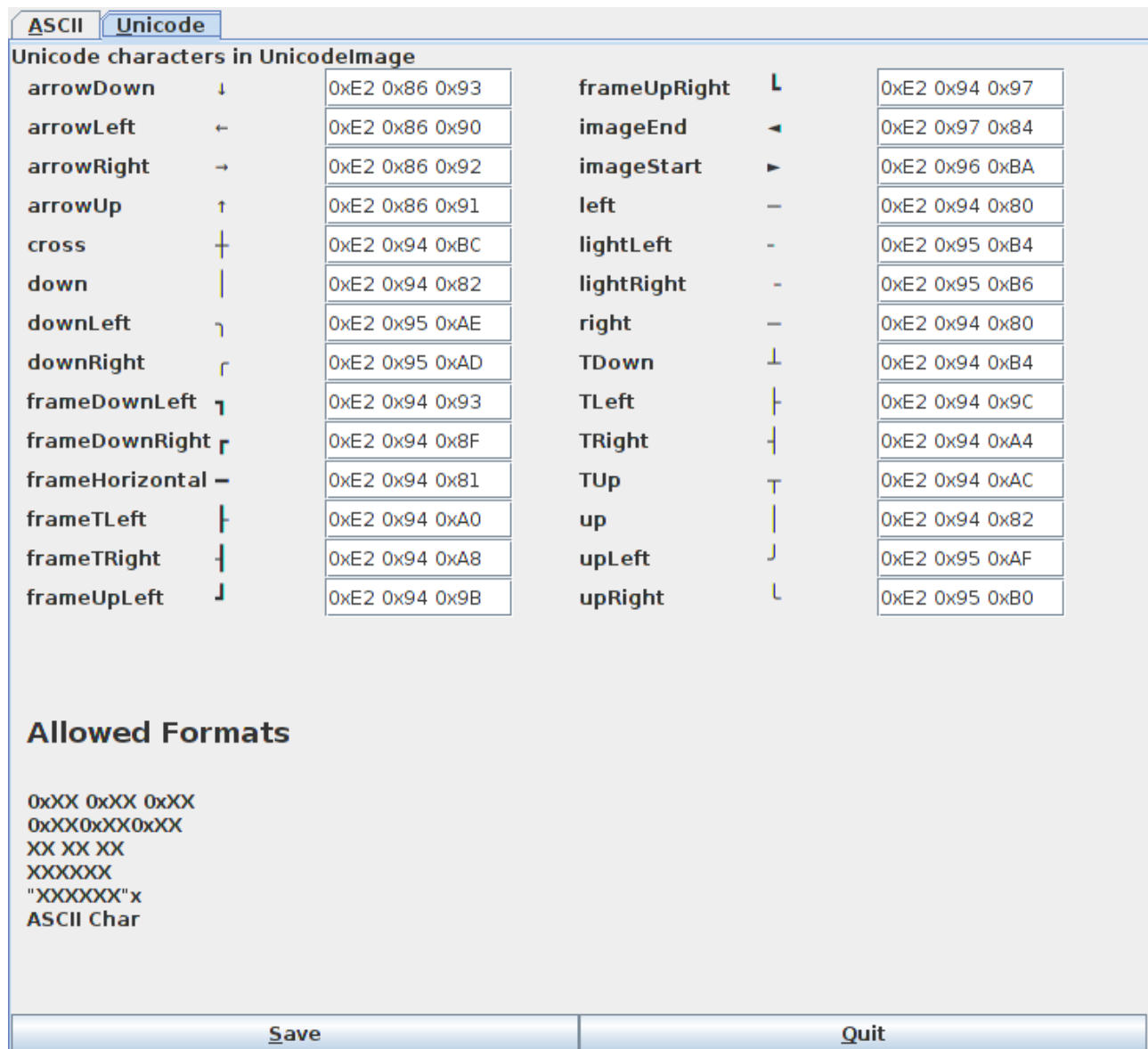


Figure 16: Drawing-symbols

As the name implies, the ASCII tab does only accept ASCII values in the textfield, whereas the Unicode tab allows the user either to enter ASCII values or the UTF-8 value of an Unicode character¹⁴. Figure 17 shows the supported encoding for the character **a**.

¹⁴Both Windows and Linux (at least all distributions with GNOME), have integrated mapping programs for Unicode characters. Windows contains the program **character map** and GNOME the program **gucharmap**. [WinXPDoc], [GuCharMap]. Both can be used to find the correct UTF-8 coding for all available fonts.

Allowed typings for a
0xEF 0xBD 0x81
0xEF0xBD0x81
EF BD 81
EFBD81
"EFBD81"x
a (only possible with ASCII characters)

Figure 17: Allowed UTF-8 codings

If the given value is not valid, the preview label on the left side of the textfield will display **invalid**.

2.7.5 Export Options

The **Exports Options** configure the export options for all BNF4OOo paragraphs (except the error paragraph). The export itself is done by running **OOoImageExport.rex** (see 2.6.6, page 54). Figure 18 shows the activated frame.

Export Options

General Options

Export Directory: /home/franz/Desktop/BNF400o-diagrams **Browse**

Prefixes for Filenames: ☒ Date ☒ Time ☒ User

Use Subfolders: ☒

BNF-Folder: BNF XML-Folder: XML

Diagram-Folder: diagram Picture-Folder: diagram-picture

BNF Options

Activate Export: ☒

Filename: BNF

File extension: bnf

XML Options

Activate Export: ☒

Filename: title tag ▼

Diagram as Text Options

Activate Export: ☒

Filename: diagram name ▼

File extension: txt

Diagram as Pic Options

Activate Export: ☒

Font Size: 20

Name: diagram name ▼

File extension: png ▼

Save **Quit**

Figure 18: Export Options

The Export Options are divided into General Options, BNF Options, XML Options, Diagram as Text Options and Diagram as Pic Options.

General Options

The General Options panel contains settings which account for all exported paragraph styles in sWriter. In the first line the user can choose the preferred Export Directory. By default it is the user's home directory.

The Prefixes for Filenames checkboxes give the user the opportunity to add date, time, and the user to the names of the exported files automatically. This feature helps to arrange the generated files, if activated. Each prefix can be chosen se-

perately by checking the proper box. In the file's name all activated elements are connected with an underline character. When all options are selected the file `image.png` would become `yyyy-mm-dd_hh:mm:ss_user_image.png`.

If the `Use Subfolders` checkbox is activated, each exported paragraph style will be exported in a separate subfolder. The names of the subfolders can be typed in the textfields below the checkbox (`BNF-Folder` for BNF, `XML-Folder` for XML, `Diagram-Folder` for diagrams exported as text and `Picture-Folder` for diagrams exported as pictures).

BNF Options

By checking the `Activate Export` checkbox all `BNF` paragraphs in the `sWriter` document will be exported when running `OooImageExport.rex`. In the `Filename` and the `File extensions` textfields the desired name and file extension can be typed. In contrast to the `XML` and the `diagram` paragraphs, the name cannot be derived directly from the paragraph. The first defined element in the BNF paragraph does not need to be the root element. Therefore the file's name has to be entered into a textfield.

If the folder already contains a file with that name, a running number will be added to the exported file's name.

XML Options

The XML-export is activated by checking the `Acivate Export` checkbox. The file name can be selected. The options are either to take the content of the XML's `title tag` or simply a `running number`. If such file already exists in the `Export Directory`, a running number is added to the name automatically. The file's extension is always `.XML`.

Diagram as Text Options

By Activating the `Activate Export` checkbox the file's name and extension can be choosen. The options for the filename are either the `diagram name` or a `running number`. If such file already exists in the `Export Directory`, a running number is added to the name automatically. The extension can be chosen freely by typing it into the textfield.

Diagram as Pic Options

If the export function is activated by selecting **Activate Export**, the user can set the **Font Size**, the file's **Name** and the graphic **Format**. The picture's size is determined indirectly by setting the font size. The bigger the font size is set, the bigger becomes the exported picture. The default value is 20. As in the **Diagram as Text Options**, the name can be either the **diagram name** or a **running number**. If such file already exists in the **Export Directory**, a running number is added to the name automatically. In the format combobox the exported picture format can be chosen from all formats available in OOo.

2.8 BNF4Shell

Nearly all functions of BNF4OOo are not only available through the OpenOffice interface but can be accessed in a **cmd/shell** as well. This way other programs can start BNF4OOo and work with the program's output from the command line. The only main functions that are not available are the **BNF2All** script and the **Diagram-pic export** script for it uses the **GraphicExportFilter** service from the OpenOffice API. [OOoAPI]

BNF4Shell is started by running **BNF4Shell.rex** in the install directory. If the option **-help** is given as a parameter, all available options will be displayed. All options are optional and not case sensitive except for the file paths on linux.

```
franz@newerubuntu:~/workspace/BNF400o$ rexx BNF4Shell.rex -help
known options are:
-BNFDIALECT filepath                (default: selection in OptionGUI.rex)
-DIAGRAM "ASCII"|"UNICODE"|"BOXEDUNICODE" (default: selection in OptionGUI.rex)
-INPUT filepath                    (default: cmd/shell)
-LOGLEVEL "TRACE"|"DEBUG"|"INFO"|"WARN"|"ERROR"|"FATAL" (default: WARN)
-MODE "BNF2DIAGRAM"|"BNF2XML"|"XML2BNF" (default: BNF2DIAGRAM)
-OUTPUT filepath                  (default: cmd/shell)
franz@newerubuntu:~/workspace/BNF400o$
```

Figure 19: **BNF4Shell.rex** options

Only the first character of every option must be typed. All following characters are optional (e.g instead of **-BNFDIALECT** it is sufficient to type **-B** or **-b**).

BNFDIALECT

This option controls the BNF-dialect used in the script. The argument must be a fully qualified or relative path to the proper XML-file. All natively supported dialects are stored in the `BNF-dialects` folder. Only versions stored in the BNF-dialects folder are listed in the `OptionGUI`. If no option is given the default BNF-dialect from the `OptionGUI` will be used (the default BNF-dialect is stored in the text file `BNF4OOo.properties`).

DIAGRAM

With the diagram option the preferred diagram type can be selected by writing the name as an argument. The user can choose between `ASCII`, `unicode` or `boxedUnicode` (see 2.4, page 41). Only the bold characters need to be typed, the rest of the word is optional. If no option is selected, the default diagram type as seen with the `OptionGUI` will be used.

INPUT

Here a fully qualified or relative path to the file containing the input is necessary as an argument. If no input option is given, BNF4Shell uses the `cmd/shell` as the input source and asks the user to enter his BNF-rules.

LOGLEVEL

Log4rexx, a logging tool written by Rony G. Flatscher¹⁵, is embedded in the code of all BNF4OOo scripts. The loglevel can be chosen by selecting `trace`, `debug`, `warn`, `error` or `fatal`. Only the bold characters must be typed, the rest of the word is optional. The default logging level is `warn`.

MODE

The available modes are `BNF2DIAGRAM`, `BNF2XML` and `XML2BNF` (see 2.6.1, page 52). Only the bold characters must be typed, the rest of the word is optional.

OUTPUT

By default BNF4Shell writes the program's to the screen (`stdout`). By adding the output option with a full qualified or relative path to the file, the output will be streamed to the given location.

¹⁵ C.f. [log4rexx].

3 BNF4OOo's Internal Structure and Logic

The following chapters give a brief overview over BNF4OOo's internal structure and logic. Although BNF4OOo aims to be flexible out of the box by allowing the user to create own BNF-dialects with customized symbols and full control over the character sets used for printing, it may not fit for all demands. Therefore this chapter gives some hints how to enhance BNF4OOo as well.

Figure 20 gives a schematic overview over the workflow in the BNF4OOo functions.

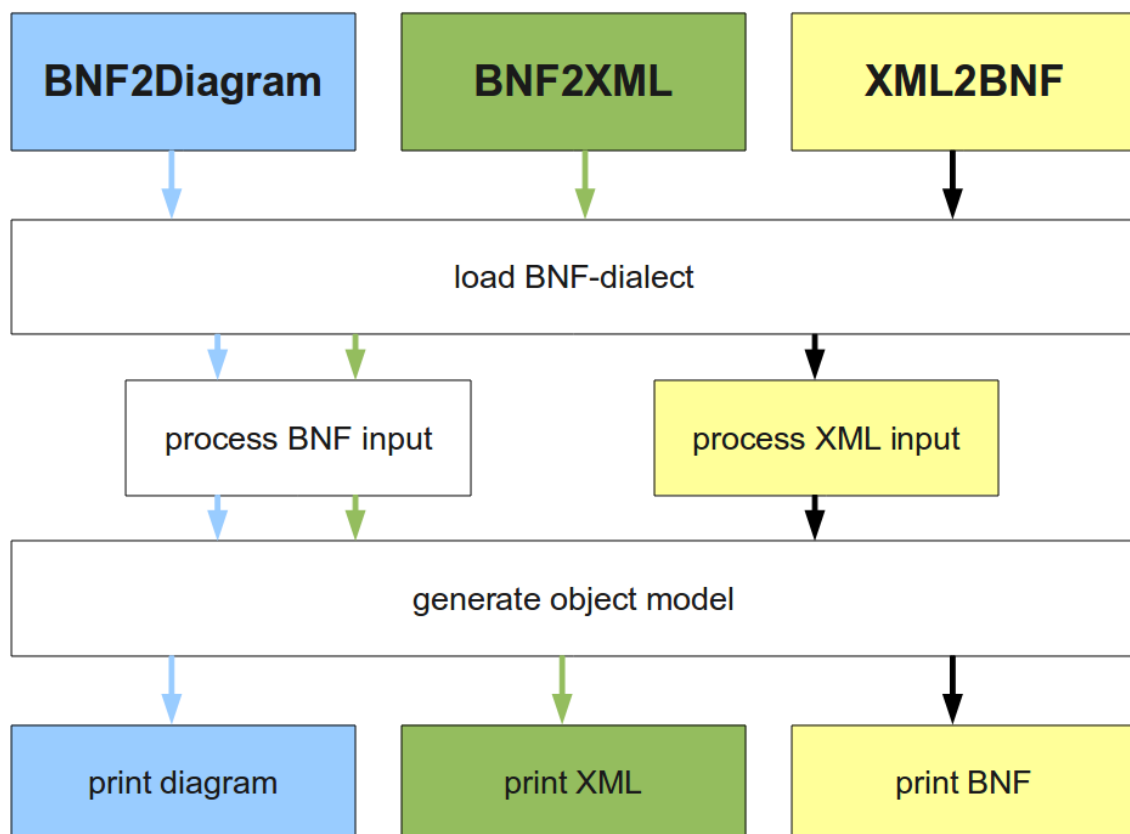


Figure 20: BNF4OOo function's workflow

The workflows of all `BNF2Diagram`, `BNF2XML` and `XML2BNF` have basically the same steps. The used BNF-dialect is loaded, the input is processed, the object model is created and the output format is generated. The following sections give a more detailed view on these steps.

3.1 Loading the BNF-dialects

All of BNF4OOo's main functions start with loading and verifying the used BNF-dialect with all its instructions. These sets of instructions are stored inside `XML-files` in the subfolder `BNF-dialects`.

XML 7 shows the BNF4OOo definition of the first BNF used in ALGOL 60.

```
<?XML version="1.0" encoding="UTF-8"?>
<BNF400oInstructionset>
  <definition symbol="::="/>
  <instruction type="nonTerminal">
    <position>circumfix</position>
    <symbol1>&lt;</symbol1>
    <symbol2>&gt;</symbol2>
  </instruction>
  <instruction type="xor" long="true">
    <position>infix</position>
    <symbol1>|</symbol1>
  </instruction>
</BNF400oInstructionset>
```

XML 7: BNF-ALGOL_60.XML

These files are the source for the BNF-dialects editor in the `Option GUI` (see chapter 2.7.3, page 60). This simple BNF-dialect has only three defined instructions, the `definition-instruction`, `Nonterminal-instructions`, and `xor-instructions`.

The `definition-instruction` is an exception from the other instructions because its position is clear and therefore must not be defined (see 2.3.1, 26).

The attribute `long` used in the instruction with `type=xor` is a special attribute. It is only interesting for instructions with the position `infix`. and since this feature may never be relevant to users, it is not editable in the `OptionGUI`. If a BNF-dialect is generated by using the `OptionGUI`, BNF4OOo automatically sets `long=true` for `XOR-instructions` with the position `infix` and `long=false` for all other `BNF-instructions`.

If an instruction with the position `infix` has the attribute `long=true` in the `XML-file`, the scope of this instruction does not end at the next element. In BNF 47, using `BNF-ALGOL_60.XML` from XML 7, `A` is defined as either `B` and `C` or `D` and `E` as shown in SD 36. Without `long=true` the meaning would be `A` is `B` followed by either `C` or `D` followed by `E` as shown in SD 37.

```
A ::= B C | D E
```

BNF 47: A BNF-rule containing a **XOR**



SD 36: The diagram generated from BNF 47 with **long=true**



SD 37: The diagram generated from BNF 47 without **long=true**

The XML file is parsed by an object of the **.BNFInstructionParser** class (source in **XMLClasses.cls**). The parser checks the instruction type, generates a fitting object from the correlating **.BNFInstruction** subclass and collects it in an object from the **.instructionCollections** class. The **.instructionCollections** class contains collections of all instructions with methods to access them by category (source in **instructionCollections.cls**).

The **.Instruction** subclasses contain all relevant information about the certain instruction and are responsible for creating the corresponding object from the **.BNFClass** subclasses (see figure 26, page 80).

```
::class Instruction public

::method type      attribute /*          the type of this instruction*/
::method position  attribute /*          position for symbols*/
::method symbol1   attribute /* the used symbol or start symbol if pos=infix*/
::method symbol2   attribute /*          the end symbol if position is infix*/
::method long      attribut /*boolean for infix instructions with long range*/

::method init /*  the attribute type is filled by init method of subclasses*/
  use arg self~position, self~symbol1, self~symbol2, self~long

::method makeObject /*          creates the correlating .BNFClass object*/
  use arg lineArray /* an array containing the affected elements of BNF-rule*/
  /*this method is implemented by subclass*/

::method toString /*          for debugging purposes*/
```

Code 1: Methods of the **.Instruction** class

Code 1 shows the methods of `.Instruction`. The most important method `makeObject` is implemented by its subclasses. Code 2 shows `.Instruction_group` as an example for the implementation of this method.

```
::class Instruction_group subclass instruction public

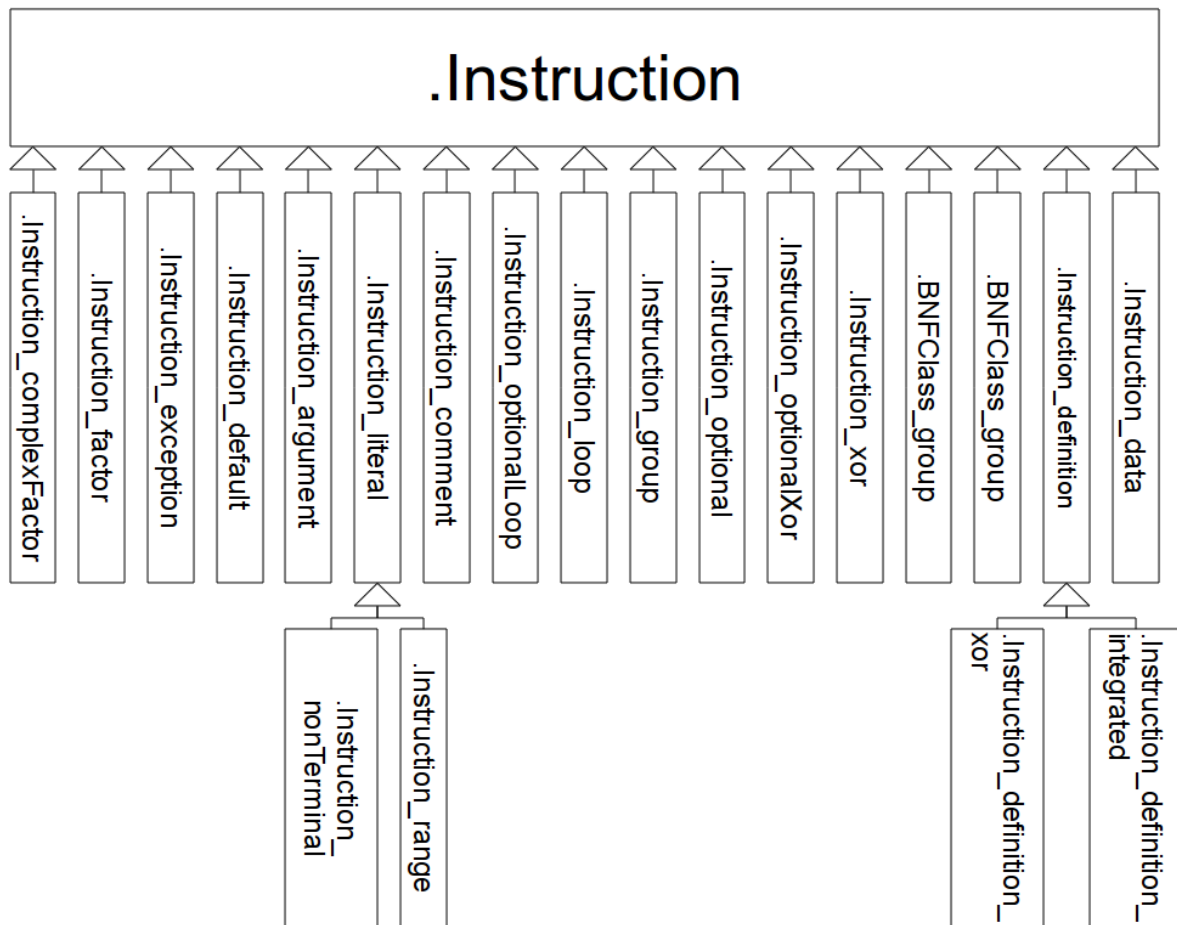
::method init
  self~init:super(arg(1),arg(2),arg(3),arg(4))
  self~type="group"

::method makeObject
  content=arg(1) /* contains all elements inside the group*/

  obj=.BNFClass_group~new
  do i over content
    if i~isInstanceOf(.BNFClass) then obj~content~append(i) /* already a BNFClass*/
    else do
      newObj=.BNFClass_data~new(i) /* since it is not a BNFClass yet-> it is data*/
      obj~content~append(newObj)
    end
  end
  return obj
```

Code 2: The `.Instruction_group` class

Figure 21 gives an overview on the implemented subclasses of `.Instruction`.

Figure 21: `.Instruction` and its subclasses

When enhancing BNF4OOo with new types of instructions, the deserved `.Instruction` subclass must be coded in `BNFInstructions.cls`. In addition the `.BNFInstructionParser` class must know the new class, otherwise no corresponding `.BNFClass` object can be created.

Code 3 shows how the `.Instructions` are connected with the Parser.

```

/*-----*/
/* Method: end_element */
/* Description: called when an end element tag has been encountered. */
/* Arguments: an XMLchunk instance. */
/*-----*/
::method end_element
  expose type position symbol symbol1 symbol2 long instrDirs l errlineidx errcharidx
  use arg chunk
  signal on user bnf4oooerror name error1
  instr=""
  if chunk~tag="instruction" then do
    select
    when type="xor" then instr=.instruction_xor~new(position,symbol1,symbol2,long)
    when type="optionalxor" then instr=.instruction_optionalXor~new(position,symbol1,symbol2,long)
    when type="optional" then instr=.instruction_optional~new(position,symbol1,symbol2,long)
    /*here are al other instructions listed as well*/
    otherwise do
      l~error("Don't know any instruction with type='"type||"'.")
      raise user bnf4oooerror
    end
  end
  /* instrDirs is the name of the instructionCollections object*/
  instrDirs~addToAllInstrDir(instr)
  /*...*/
end
return

```

Code 3: The `.BNFInstructionParser`'s `end_element` method

This method is called when a closing tag is parsed. The parser checks the parsed `type` attribute value and creates the correlating instruction object. A `new_instruction` object is introduced by inserting the line shown in Code 4.

```

when type="new_instruction" then instr=.new_instruction~new(position,symbol1,symbol2,long)

```

Code 4: Creating a `new_instruction` object in `.BNFInstructionparser`

3.2 The Input and Output System

The input and output system is the only major difference between BNF4OOo and `BNF4Shell.rex`. Figure 22 gives a graphic overview over the scripts involved in both approaches.

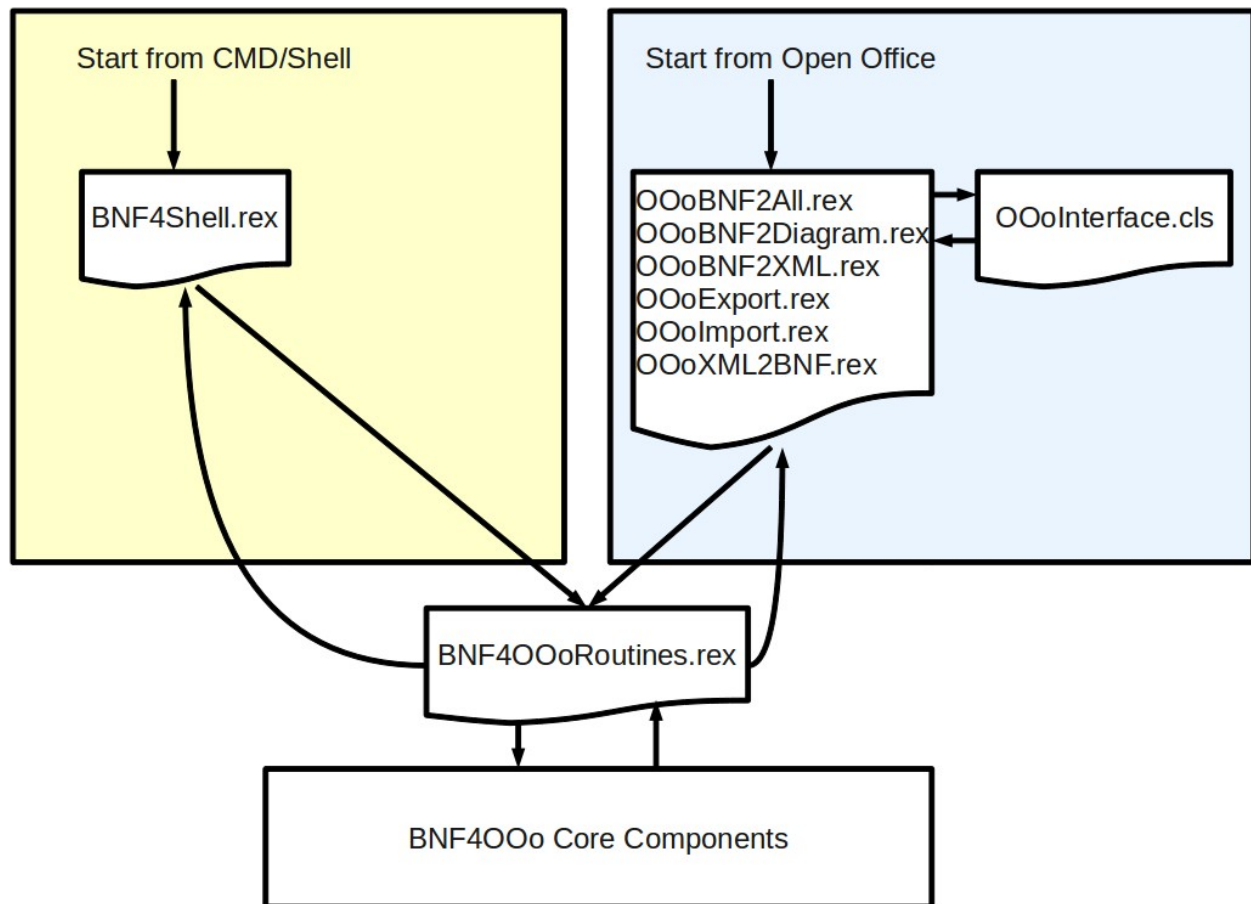


Figure 22: Schematic overview of BNF4OOo and `BNF4Shell.rex`

The structural simpler approach is the one of `BNF4Shell.rex`. The script itself deals with all different modes, diagram types etc. All necessary information is collected and sent to the `BNF4Routines.rex` routines which take the role of an interface between the input/output scripts and the BNF4OOo core components. The core components transfer the data (e.g. a BNF-dialect into a rail diagram), sends it back to `BNF4OOoRoutines.rex` which passes it to `BNF4Shell.rex`. The input and output of `BNF4Shell.rex` is described in detail in 2.8, page 67.

Basically the OOo scripts work with the same schema as `BNF4Shell.rex` with only two main differences. First, all the functionality is not bundled in a single script. Since a click on a macro button activates a script without any arguments, each function of BNF4OOo has its own script. Second, all routines for writing, editing and reading the `sWriter` document are neither in the scripts itself, nor in `BNF4OOoRoutines.rex` but in the class `.OOoInterface` in `OOoInterface.cls`.

Figure 23 and figure 24 give an overview over the most important input and output methods of `OoInterface.rex`.

Input methods	Description
<code>getAllParagraphs (paragraphstyle name)</code>	Returns an array containing all paragraphs of the given paragraphstyle in the <code>sWriter</code> document.
<code>getInputFromCurrentSelection ()</code>	Returns an array containing all selected paragraphs.
<code>getInputFromStart (paragraphstyle name)</code>	Returns an array containing the first found paragraph of the given paragraphstyle in the <code>sWriter</code> document.
<code>getInputFromTextcursor (paragraphstyle name)</code>	Returns an array containing the first found paragraph of the given paragraphstyle. The search starts from the current <code>OoInterface</code> textcursor position.

Figure 23: Important input and output methods of `OoInterface.rex`

The paragraphstyle name can either be BNF, XML or diagram. The method `getAllParagraphs` is used by the `Export` script, e.g. to process all diagrams. The method `getInputFromCurrentSelection` is used when the range is set to selection (see chapter 2.7.1, page 56). If all paragraphs of one style (e.g. BNF) are transferred into another paragraph (e.g. diagram), `getInputFromStart` is used until all input paragraphs in the `sWriter` document are replaced by output paragraphs. The script `OoBNF2All.rex` uses the method `getInputFromTextcursor` to jump through the document since it does not replace the input paragraphs but add the output paragraphs below the input paragraphs.

Output	Description
<code>printBNF (BNFArray)</code>	Prints <code>BNFArray</code> at the current <code>OoInterface</code> textcursor position.
<code>printDiagram (imageArray)</code>	Prints <code>imageArray</code> at the current <code>OoInterface</code> textcursor position.
<code>printError (errorArray)</code>	Prints <code>errorArray</code> at the current <code>OoInterface</code> textcursor position.
<code>printXML (XMLArray)</code>	Prints <code>XMLArray</code> at the current <code>OoInterface</code> textcursor position.

Figure 24: Important input and output methods of `OoInterface.rex`

The output methods create a new paragraph of the corresponding style and print the output array. The `BNFArray`, `errorArray` and `XMLArray` are simple arrays containing the paragraph's lines. The `imageArray` on the other hand can contain more than one diagram. Therefore, the `imageArray` itself contains arrays with the diagram lines and a new paragraph is created for every single diagram.

3.3 Generating the Object Model

In each main function in BNF4OOo an object model is created. In `BNF2Diagram` and `BNF2XML` the object model is generated by the `.BNFController` class by using the `.Instruction` classes (see 3.1, page 70). In `XML2BNF` the object model is generated directly in the `.DITAParser` class (stored in `XMLClasses.cls`) while parsing the XML content as seen in Code 5.

```

/*-----*/
/* Method: start_element                                     */
/* Description: called when a start element tag has been encountered. */
/* Arguments:  an xmlchunk instance.                         */
/*-----*/
::method start_element private
  expose openTags tagArray objectArray rootTag instrDirs 1 argument
  use arg chunk

  if chunk~tag=rootTag then return /* no need to parse that tag*/
  select
    when chunk~tag="title"      then obj=.BNFClass_definition~new
    when chunk~tag="groupseq"    then obj=.BNFClass_group~new
    when chunk~tag="groupchoice" then obj=.BNFClass_xor~new
  /*...*/
end
/*...*/

```

Code 5: Generating the object model in `XML2BNF`

The root of the object model is an array containing all `.BNFClass_definition` objects which are not part of another definition. These `.BNFClass_definition` objects contain all `.BNF_Class` objects which are part of their definition. Figure 25 illustrates the object tree created by BNF 48 using the XML-EBNF dialect.

```

element1 ::= sub1 sub2+
element2 ::= sub3

```

BNF 48: Two unrelated BNF-rules

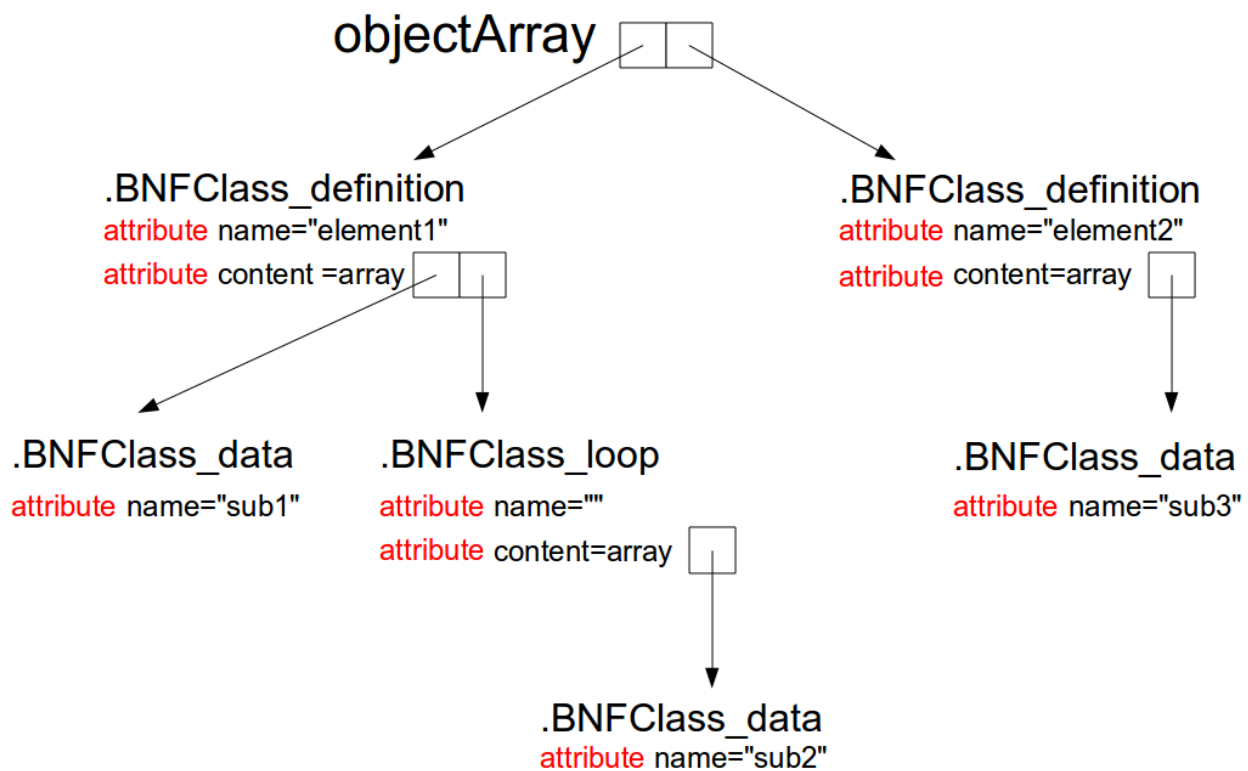


Figure 25: The object tree generated out of BNF 48

The `.BNFClass` class, like the `.Instruction` class, contains methods which are implemented by its subclasses. Code 6 shows the most important attributes of `.BNFClass` and all methods that need to be implemented by its subclasses.

```

::class BNFCClass public

::method content attribute /*      an array with all containing BNF-Objects*/
::method name attribute /*      data, nonTerminals, etc.. have names*/
::method type attribute /*      type of Object (i.e. "xor")*/
::method DITATYPE attribute /*      DITA Tag name if it has one*/
::method DITAAttr attribute /*      DITA attribute name and value if it has one*/
::method l attribute private /*      the log4rexx logger*/

::method init
  expose l
  use strict arg name=.nil
  /*subclasses must fill attributes content, name, type, DITATYPE and DITAAttr*/

::method toImage /*      prints diagram recursively*/
  expose l
  use arg image, line
  /*image is a two dimensional array containg the printed characters*/
  /*line is the line of the image which is actually printed*/

  /*here the subclasses implement their diagram printing logic*/
return image

::method toXML public /*      prints xml recursively*/
  expose l
  use arg XMLArray, tabCount
  /*XMLArray contains the printed XML lines*/
  /*tabCount shows the nesting level*/

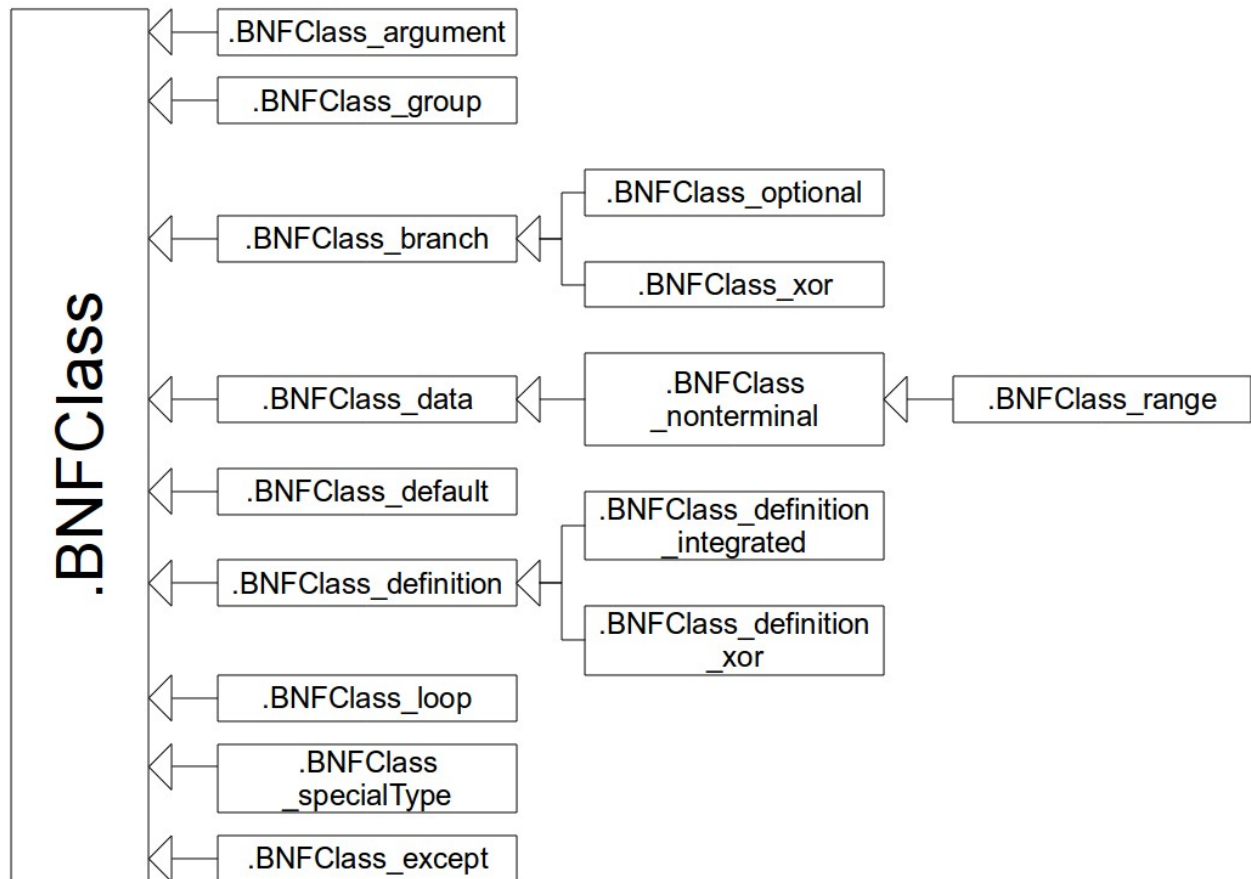
  /*here the subclasses implement their XML printing logic*/
  return XMLArray

::method getLength public /*      returns the space needed in diagram*/
  expose l
  /*here the subclasses need to return their      */
  /*space and the space demands of all contains BNFCClasses*/
return length

```

Code 6: The .BNFCClass

Figure 26 shows all subclasses of .BNFCClass that are used in BNF4OOo.

Figure 26: `.BNFClass` and its subclasses

4 Outlook

In this paper the program BNF4OOo is described. BNF4OOo gives the user the opportunity to create ASCII or Unicode character based syntax diagram out of any of the supported BNF-dialects. In addition The BNF-dialects can be transformed into, and generated out of, a DITA based XML syntax. The interface of this program is embedded in OpenOffice which provides a lot of functions for further processing with the generated diagrams and text. On the other hand all important functions of BNF4OOo can be accessed through a windows command line or a Linux/Mac shell as well using BNF4Shell.

The functionality of BNF4OOo is useful for programmers who want to document their functions, routines methods and interfaces with automatically generated railroad diagrams. Even when no actual BNF-dialect fits their need perfectly they can create their own customized BNF-dialect with their own customized syntax.

The next step for BNF4OOo's functionality is to enable a conversion of all available BNF-dialects into the original Backus Naur Form which is not possible right now. In addition new features for the Export functions (e.g. transparent background for `.png` files) will be implemented.

5 References

Bibliography

- [ABNF01] Network Working Group: 2008, Request for Comments 5234, Augmented BNF for Syntax Specifications: ABNF, Title Page, <ftp://ftp.rfc-editor.org/in-notes/std/std68.txt>, retrieved on 31.03.2011.
- [ABNF02] Network Working Group: 2008, Request for Comments 5234, Augmented BNF for Syntax Specifications: ABNF, Page 3, <ftp://ftp.rfc-editor.org/in-notes/std/std68.txt>, retrieved on 31.03.2011.
- [ABNF03] Network Working Group: 2008, Request for Comments 5234, Augmented BNF for Syntax Specifications: ABNF, Page 4, <ftp://ftp.rfc-editor.org/in-notes/std/std68.txt>, retrieved on 31.03.2011.
- [ABNF04] Network Working Group: 2008, Request for Comments 5234, Augmented BNF for Syntax Specifications: ABNF, Page 5, <ftp://ftp.rfc-editor.org/in-notes/std/std68.txt>, retrieved on 31.03.2011.
- [ABNF05] Network Working Group: 2008, Request for Comments 5234, Augmented BNF for Syntax Specifications: ABNF, Page 6, <ftp://ftp.rfc-editor.org/in-notes/std/std68.txt>, retrieved on 31.03.2011.
- [ABNF06] Network Working Group: 2008, Request for Comments 5234, Augmented BNF for Syntax Specifications: ABNF, Page 7, <ftp://ftp.rfc-editor.org/in-notes/std/std68.txt>, retrieved on 31.03.2011.
- [ABNF07] Network Working Group: 2008, Request for Comments 5234, Augmented BNF for Syntax Specifications: ABNF, Page 8, <ftp://ftp.rfc-editor.org/in-notes/std/std68.txt>, retrieved on 31.03.2011.
- [AFHMMPa] W. David Ashley, Rony G. Flatscher, Mark Hessling, Rick McGuire, Mark Miesfeld, Lee Peedin, Rainer Tammer and Jon Wolfers, Open Object Rexx Reference 4.0.0, How to Read the Syntax Diagrams page 35-36, retrieved on 31.03.2011.

- [ALGOL] Revised Report on the Algorithmic Language Algol 60, Revised Report on the Algorithmic Language Algol 60, <http://www.masswerk.at/algol60/report.htm>, retrieved on 31.03.2011.
- [BrauFra] Frank Braun, Generation of Syntax Diagrams, <http://www-cgi.uni-regensburg.de/~brf09510/syntax.html>, retrieved on 31.03.2011.
- [BSF4Re] Rony G. Flatscher: BSF4ooRexx, <http://wi.wu.ac.at/rgf/rexx/bsf4oorexx/archive/2011/2011-03-27/>, retrieved on 31.03.2011.
- [DITA01] Introduction to the Darwin Information Typing Architecture, <http://www.ibm.com/developerworks/XML/library/x-dita1/>, retrieved on 31.03.2011.
- [DITADoc01] The DITA documentation, <http://docs.oasis-open.org/dita/v1.1/OS/langspec/langref/syntaxdiagram.html>, retrieved on 31.03.2011.
- [DITADoc02] The DITA documentation, <http://docs.oasis-open.org/dita/v1.1/OS/langspec/langref/title.html>, retrieved on 31.03.2011.
- [DITADoc03] The DITA documentation, <http://docs.oasis-open.org/dita/v1.1/OS/langspec/langref/groupseq.html>, retrieved on 31.03.2011.
- [DITADoc04] The DITA documentation, <http://docs.oasis-open.org/dita/v1.1/OS/langspec/langref/groupchoice.html>, retrieved on 31.03.2011.
- [DITADoc05] The DITA documentation, <http://docs.oasis-open.org/dita/v1.1/OS/langspec/langref/var.html>, retrieved on 31.03.2011.
- [DITADoc06] The DITA documentation, <http://docs.oasis-open.org/dita/v1.1/OS/langspec/langref/repsep.html>, retrieved on 31.03.2011.
- [DITADoc07] The DITA documentation, <http://docs.oasis-open.org/dita/v1.0/langspec/fragref.html>, retrieved on 31.03.2011.

- [Eclipse] Eclipse IDE Classic, <http://www.eclipse.org/downloads/>, retrieved on 31.03.2011.
- [FryJos01] Josef Frysak: Automating OpenOffice – ooRexx Nutshells, Chapter(s) 4.2, Page(s) 21, http://wi.wu-wien.ac.at/rgf/diplomarbeiten/BakkStuff/2008/200809_Frysak/200809_Frysak_Automating_OOo_ooRexx_Nutshells.pdf, retrieved on 31.03.2011.
- [FryJos02] Josef Frysak: Automating OpenOffice – ooRexx Nutshells, Chapter 6.1.4, Page 55, http://wi.wu-wien.ac.at/rgf/diplomarbeiten/BakkStuff/2008/200809_Frysak/200809_Frysak_Automating_OOo_ooRexx_Nutshells.pdf, retrieved on 31.03.2011.
- [GuCharMap] GNOME program gucharmap, <http://live.gnome.org/Gucharmap>, retrieved on 31.03.2011.
- [ISOEBNF01] ISO/IEC 14977: 1996 (E), Page vi, [http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996\(E\).zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip), retrieved on 31.03.2011.
- [ISOEBNF02] ISO/IEC 14977: 1996 (E), Page 2,7, [http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996\(E\).zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip), retrieved on 31.03.2011.
- [JavaJDK] Java Developing Kit, <http://www.oracle.com/technetwork/java/javase/downloads/index.html>, retrieved on 31.03.2011.
- [Knuth] Donald E. Knuth: Communications of the ACM, 1964, Volume 7, Number 12, Page(s) 735, 736, <http://portal.acm.org/citation.cfm?doid=355588.365140>, retrieved on 31.03.2011.
- [log4rexx] Rony G. Flatscher: log4rexx, <http://wi.wu.ac.at/rgf/rexx/orx18/log4r/>, retrieved on 31.03.2011.
- [OOo] OpenOffice.org, <http://www.openoffice.org/>, retrieved on 31.03.2011.
- [OOoAPI] OpenOffice.org, API – interface XStorable, 2010, <http://api.openoffice.org/docs/common/ref/com/sun/star/drawing/Graphic-ExportFilter.html>, retrieved on 31.03.2011.

- [ooRexx] open object Rexx, <http://www.oorexx.org/>, retrieved on 31.03.2011.
- [RailRoad] Railroad Diagram Generator, <http://railroad.my28msec.com/rr/ui>, retrieved on 31.03.2011.
- [SchoDo] Stefan Schoergenheimer, Markus Dopler, EBNF Visualizer, <http://dotnet.jku.at/applications/Visualizer/>, retrieved on 31.03.2011.
- [KriTam] Tamás Krisztin : Rexx plugin for Eclipse, <http://sourceforge.net/projects/rexxeclipse/>, retrieved on 31.03.2011.
- [SouForge] SourceForge.net, <https://sourceforge.net/about>, retrieved on 31.03.2011
- [ThiePe] Peter Thiemann, Ebnf2ps: Peter's Syntax Diagram Drawing Tool, <http://www.informatik.uni-freiburg.de/~thiemann/haskell/ebnf2ps/>, retrieved on 31.03.2011.
- [UMIFOO] Andrew Pitonyak: Useful Macro Information For OpenOffice, Chapter(s) 5.29.2, Page 86, <http://www.pitonyak.org/AndrewMacro.odt>, retrieved on 31.03.2011.
- [W3CXML] The World Wide Web Consortium, Extensible Markup Language (XML) 1.0 (Fifth Edition) W3C Recommendation 26 November 2008, <http://www.w3.org/TR/REC-XML/>, retrieved on 31.03.2011.
- [Wikipe01] The Wikipedia website, http://en.wikipedia.org/w/index.php?title=Backus%E2%80%93Naur_Form&oldid=416893530, retrieved on 31.03.2011.
- [Wikipe02] The Wikipedia website, <http://en.wikipedia.org/w/index.php?title=ALGOL&oldid=419213481>, retrieved on 31.03.2011.
- [Wikipe03] The Wikipedia website, http://en.wikipedia.org/w/index.php?title=Syntax_diagram&oldid=405552927, retrieved on 31.03.2011.
- [WinXPDoc] Windows XP documentation, http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/app_charmap.mspx?mfr=true, retrieved on 31.03.2011.

- [WirthNi] Niklaus Wirth: Communications of the ACM, 1977, Volume 20, Number 11, Page 822, <http://portal.acm.org/citation.cfm?id=359863.359883>, retrieved on 31.03.2011.
- [XBNF01] California State University San Bernadino, Extreme BNF, http://www.ci.csusb.edu/dick/math/intro_ebnf.html, retrieved on 31.03.2011.
- [XMLPar] W. David Ashley: A SAX-like XML-parser, <http://oorexx.svn.sourceforge.net/viewvc/oorexx/incubator/orxutils/xml/xmlparser.cls?view=markup>, retrieved on 17.04.2011.

6 Attachment

6.1 Installation Guide

The following chapters describe the Installation of BNF4OOo with screenshots for every installation step.

6.1.1 Downloading the Components

For a proper installation of BNF4OOo all required software has to be installed on the user's system (see 1.1, page 12). The BNF4OOo project itself is developed on sourceforge, the biggest open source software developing website on the Internet with more than 230.000 projects registered. [SouForge].

The folder containing the project's files can be downloaded from https://sourceforge.net/projects/bnf4ooo/files/BNF4OOo_v.0.9.zip/download to any preferred location on the user's desktop.

6.1.2 Running the Installation Script

The script `addBNF4OOoButtons.rex` starts OpenOffice with all BNF4OOo components right from the `cmd/shell`. This way the user can get in touch with the program for the first time without installing anything on the user's system except for the BNF4OOo folder.

If BNF4OOo is supposed to be started from inside OpenOffice, the files need to be integrated in OpenOffice's file structure for macros. OpenOffice knows three ways of storing a macro, either as an openoffice.org macro, a user macro or as a document macro. [Fry-Jos01], [UMIFOO]. Integrating the scripts as user macros can be done by simply running the script `installBNF4OOo.rex` which copies all required files directly into the user's oorex scripting folder. If there is a former version of BNF4OOo installed, it will be replaced automatically.

With OpenOffice knowing about the scripts, BNF4OOo can be started by opening the `Run Macro` menu and clicking `addBNF4OOoButtons.rex`.

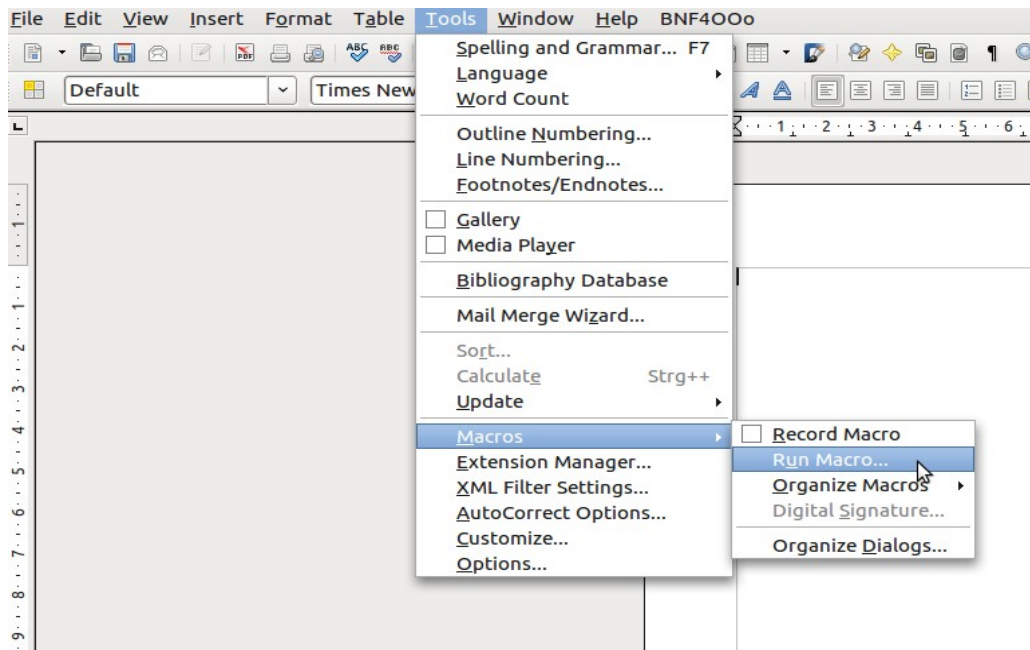


Figure 27: Running `addBNF4OOoButtons.rex` as a macro part 1

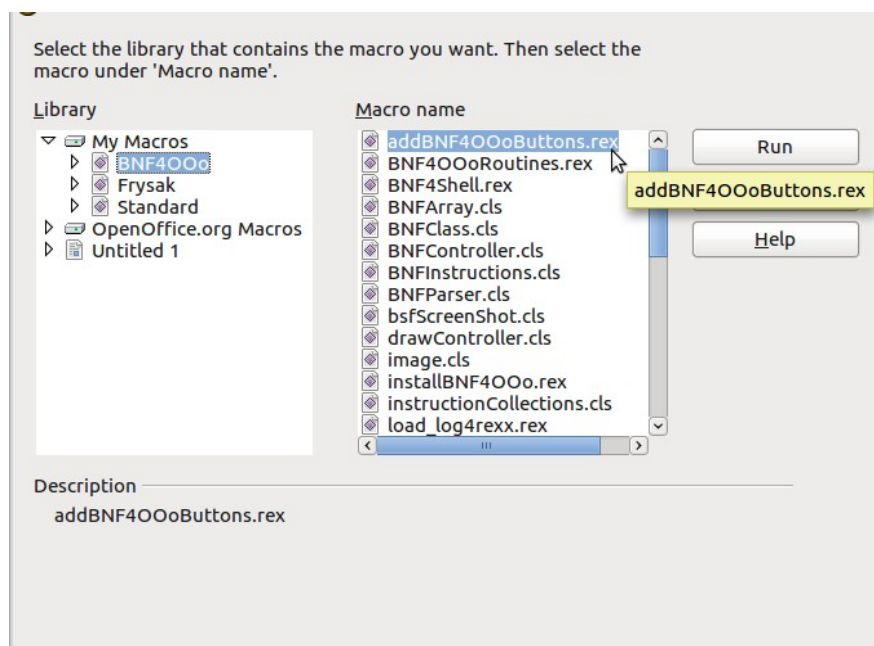


Figure 28: Running `addBNF4OOoButtons.rex` as a macro part 2

The script creates the BNF4OOo macro buttons and the paragraphs `BNF`, `XML`, `diagram` and `error` which are used for creating the different formats. To remove the changes, the macro `removeBNF4OOoButtons.rex` needs to be clicked the same way as `addBNF4OOoButtons.rex`. Unfortunately OpenOffice cannot recreate the toolbar once it has been removed in the same session. It only creates an empty toolbar which cannot be

removed. Therefore OpenOffice has to be restarted in order to run `addBNF4OOoButtons.rex` successfully. [FryJos02].

6.1.3 Integrating the Startscript in the OOo Toolbar

When working with BNF4OOo on a regular basis, the way described above may appear very time consuming. Therefore it may be a good idea to integrate the start script directly into the `sWriter`'s menubar so it can be run with a single click.

To enter the "Customize" menu click on the `Tools` button in the swriter's menu bar, and click on `Customize` in the dropdown menu.

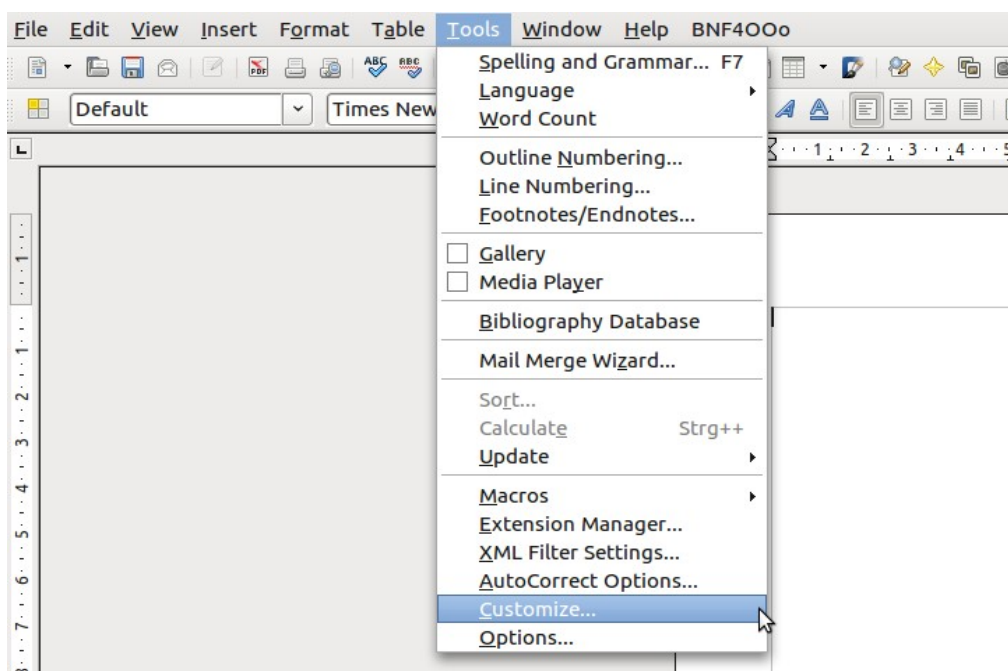


Figure 29: Entering the toolbar customize menu

Now in the appearing **Customize** dialog windows there is a dropdown box which shows all

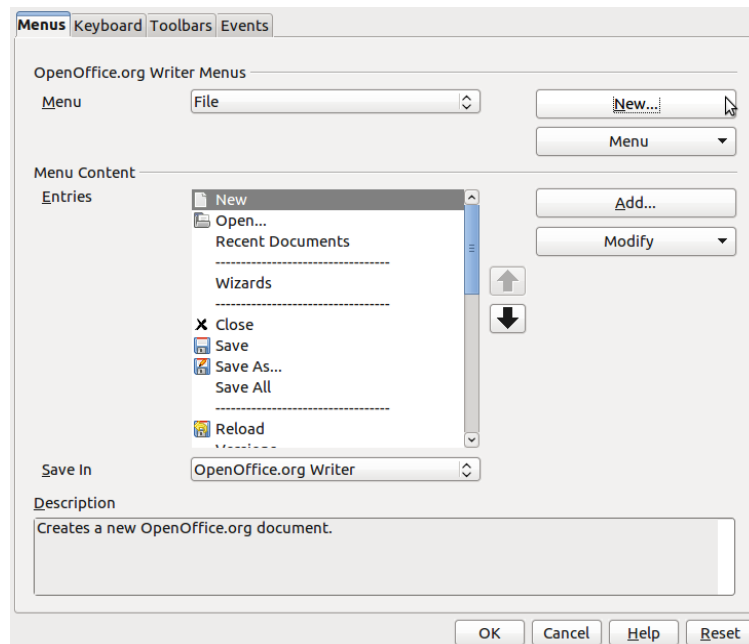


Figure 30: Creating a new menu part 1

entries in the **sWriter**'s menu bar. On the right side, click the **New** button to create an new menu entry.

For the new **Menu name** type **BNF400o**. Click **OK** to save the entry and close the dialog window.

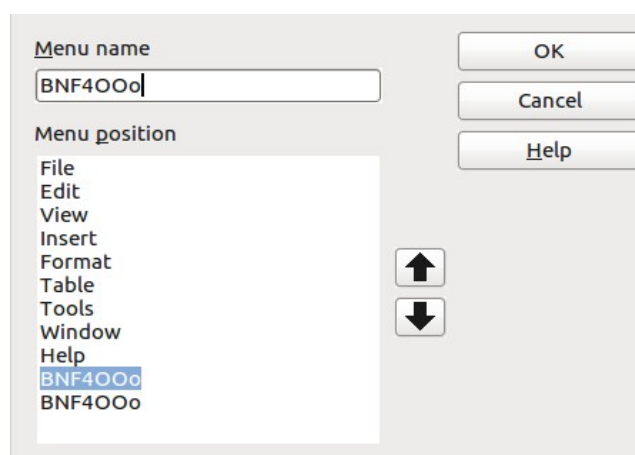


Figure 31: Creating a new menu part 2

Back in the "Customize" window, the new menu **BNF400o** can be selected from the dropdown box. By now the **Menu Content** of **BNF400o** is still empty. Change this by clicking the **Add** button on the right side.

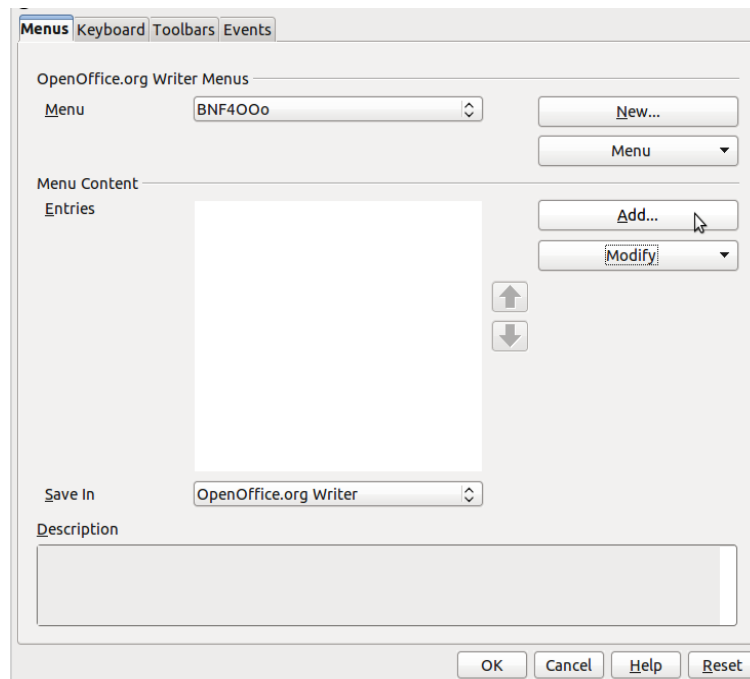
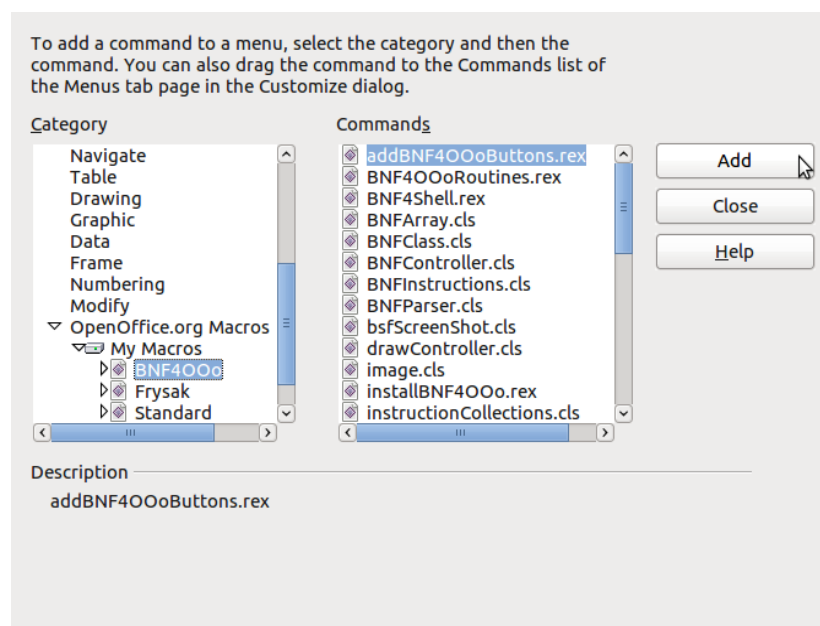


Figure 32: Adding Entries to the new menu

Once again, a new window appears which shows a list of all available **Categories** and another list with the category's **Commands**. In the left list, select **OpenOffice.org Macros**, **My Macros** and finally **BNF4OOo**. Now the right list shows all scripts of BNF4OOo. First click on the script **addBNF4OOoButtons.rex**, click **Add** and then repeat the procedure with **removeBNF4OOoButtons.rex**. After that close the window by clicking **Close**.

Figure 33: Adding **addBNF4OOoButtons.rex** to the menu

Finally, make sure that all changes are saved by closing the Customize menu with the **OK** button. The **sWriter**'s menu bar now has a new menu named BNF4OOo which contains the starting and the closing script.

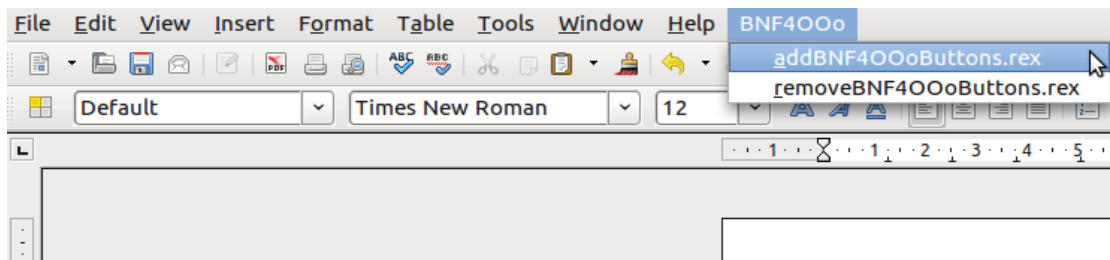


Figure 34: Running **addBNF4OOoButtons.rex** from the toolbar

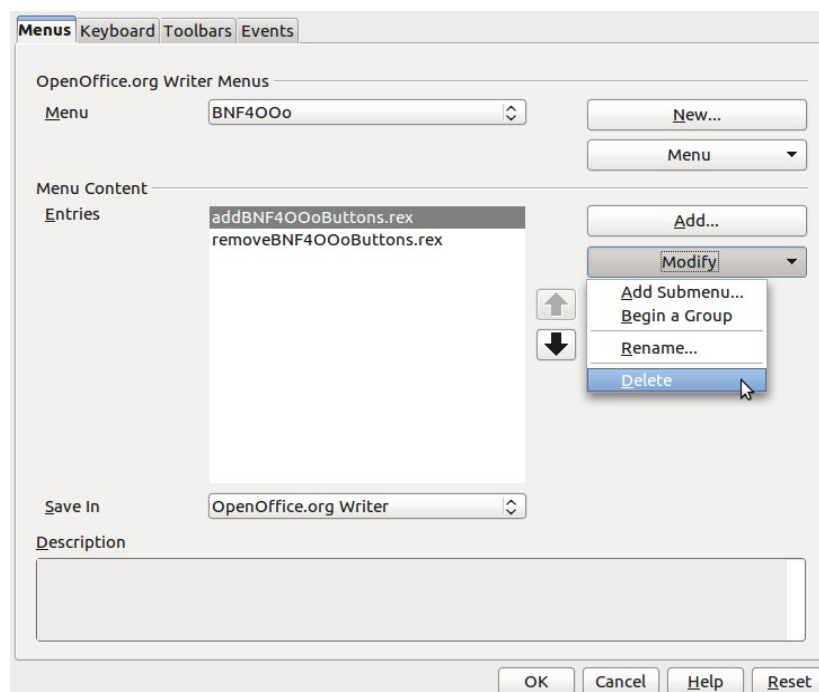


Figure 35: Removing BNF4OOo from the toolbar

To remove the BNF4OOo menu open the **Customize** menu and click **Delete** in the right menu box.

6.2 The OptionGUI – A Quick Overview

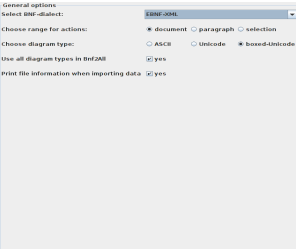
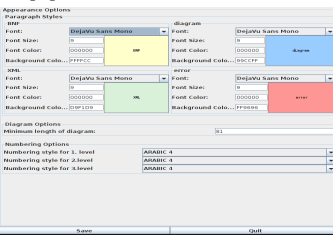
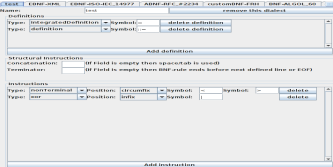

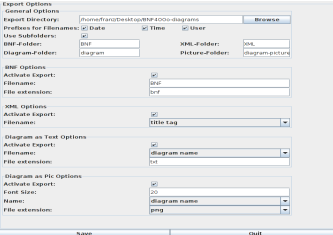
<div><h2>General Options</h2></div>	<div><h3>Select BNF-dialect</h3><p>Selects the BNF-dialect used for all BNF4OOo functions</p></div> <div><h3>Choose range for actions</h3><p>Selects if BNF4OOo transforms all paragraphs in the document, only the first paragraph or the actual selection.</p></div> <div><h3>Choose diagram type</h3><p>Selects if BNF4OOo generates ASCII images, Unicode images or boxed Unicode images.</p></div> <div><h3>Use all diagram types in BNF2All</h3><p>If activated, BNF2All will generate ASCII images, Unicode images and boxed Unicode images.</p></div>
<div><h2>Appearance</h2></div>	<div><h3>Paragraph Styles</h3><p>Changes font, font size, font color and backgroundcolor for all BNF4OOo paragraphs.</p></div> <div><h3>Diagram Options</h3><p>Sets minimium length for diagrams.</p></div> <div><h3>Numbering Options</h3><p>Sets numbering options for all headings used in BNF2All.</p></div>
<div><h2>BNF-Dialects</h2></div>	<p>Adds, edits and removes BNF-dialects from BNF4OOo. Inside the BNF-dialects menu, every BNF-instruction can be edited.</p>
<div><h2>Drawing-symbols</h2></div>	<p>Edits the UTF-8 encoded characters used to print ASCII and Unicode images.</p>
<div><h2>Export Options</h2></div>	<div><h3>General Options</h3><p>Changes the export directory, prefixes(date, time, user) and the used subfolders.</p></div> <div><h3>BNF, XML and Diagram Options</h3><p>Activates export for this paragraphs and changes the exported file's name and extension.</p></div> <div><h3>Diagram as Pic Options</h3><p>Activates export, changes font size for exported images, the picture's name and the exported file's extension.</p></div>

Figure 36: Quick overview on the BNF4OOo subfolders

6.3 The BNF4OOo Files – A Quick Overview

Figures 37, 38 and 39 give a quick overview on the source files of BNF4OOo.

Subfolders containing resources	Description
BNF-dialects	Contains all stored BNF-dialects.
BNF-samples	Some samples of different BNF-dialects.
XML	Contains the XML-schema files.

Figure 37: Quick overview on the BNF4OOo subfolders

Files	Description
<code>BNF4OOo.properties</code>	Contains all BNF4OOo settings except the drawing-symbols.
<code>BNF4OOoRoutines.rex</code>	Provides routines for BNF4Shell and BNF4OOo.
<code>BNF4Shell.rex</code>	A reduced version of BNF4OOo which runs in CMD/Shell-
<code>BNFArray.cls</code>	Contains the parsed BNF arrays.
<code>BNFClass.cls</code>	Contains the BNFClass class and all subclasses of BNFClass.
<code>BNFController.cls</code>	Controls the transformation of BNF and XML data into the object model and the other way round.
<code>BNFInstructions.cls</code>	Contains the <code>.BNFInstruction</code> class and all subclasses of <code>.BNFInstruction</code> .
<code>BNFParser.cls</code>	Parses the BNF input into a BNFArray class-
<code>bsfScreenShot.cls</code>	Contains the screenshot scripts written by Rony G. Flatscher.
<code>drawController.cls</code>	Controls the printing of the syntax diagrams.
<code>image.cls</code>	Contains the image classes.
<code>installBNF4OOo.rex</code>	The installation script for OOo.
<code>instructionCollections.cls</code>	Contains collections of the loaded <code>BNF-instructions</code> .
<code>load_log4rexx.rex</code> <code>log4rexx_appender.cls</code> <code>log4rexx.cls</code> <code>log4rexx.css</code> <code>log4rexx_filter.cls</code> <code>log4rexx_init.cls</code> <code>log4rexx_layout.cls</code> <code>log4rexx_logger.cls</code> <code>log4rexx.properties</code> <code>rgf.sockets.cls</code>	The <code>log4rexx</code> scripts and classes written by Rony G. Flatscher.

Figure 38: Quick overview on the BNF4OOo files part 1

OOoBNF2All.rex OOoBNF2Diagram.rex OOoBNF2XML.rex OOoImport.rex OOoExport.rex OooXML2BNF.rex	The BNF4OOo scripts for the menu buttons.
OOoInterface.cls	Contains the OOoInterface class which is responsible for reading and wring from and in the sWriter document.
OptionGUI.rex	BNF4OOo's GUI script.
parcel-descriptor.XML	Created automatically by installBNF4OOo.rex. This file makes OOo aware of the BNF4OOo scripts.
publicRoutines.rex	Provides a set of routines needed by most of BNF4OOo's scripts and classes.
removeBNF4OOoButtons.rex	Removes the BNF4OOo buttons from the sWriter's toolbar.
addBNF4OOoButtons.rex	Installs the BNF4OOo buttons in the sWriter's toolbar.
XMLClasses.cls	Contains the classes needed for parsing and writng XML files.
XMLparser.cls	A SAX like parser interface written by W. David Ashley. Enhanced by some minor modifications.

Figure 39: Quick overview on the BNF4OOo files part 2