

XBldr

Software Design Specification

<Sub Title>

|  |  |
| --- | --- |
| Version: | 3/10/2006 |
| Document No: |  |
| Print Date: |  |
| Release Date: |  |
| Release State: |  |
| Approval State: | Draft |
| Approved by: |  |
| Prepared by: |  |
| Reviewed by: |  |
| Path Name: |  |
| File Name: |  |
| Confidentiality Category: |  |

**Table of Contents**

Table Of Tables and Figures 2

1. Introduction 1

1.1. Purpose 1

1.2. Scope 1

1.3. Terminology 1

1.4. Overview 1

1.4.1. XLanguage 1

1.4.2. Grammar Definition 1

1.5. Text Font Usage 1

2. XBldr and XLanguage 2

2.1. XBldr 2

2.1.1. … 2

2.2. XLanguage 2

2.2.1. … 2

3. Gramma Definition 2

3.1. Gramma Definition File 2

3.1.1. … 2

3.1.2. … 2

3.2. Embedded Gramma Definition, .cs File 2

4. Source Language Implementations 2

4.1. C# Implementation 2

4.1.1. Source File embedded Gramma Definition 2

4.1.2.1. Inherited Parser Engine 2

4.2. … 2

5. Keywords 2

5.1. Options 2

5.1.1. ?Name I! 2

5.1.2. ?Class I! 3

5.1.3. ?Options ? 3

5.1.4. ?Name I! 3

5.1.5. ?Name I! 3

5.1.6. 3

5.2. Defaults 3

5.2.1. !ASCII 3

5.2.2. !UTF7 3

5.2.3. !UTF8 3

5.2.4. !UTF16 3

5.2.5. !UTF32 3

5.2.6. !Unicode 3

5.2.7. !BEUnicode 3

5.2.8. !Token 3

5.2.9. !WS 3

Appendix A 1

# Table Of Tables and Figures

[Table 1 Fonts and their usage 2](#_Toc154482308)

Figure 1 … 3

# Introduction

## Purpose

This document serves as both a User Guide and SDS for the XBldr project. It is to be used in its development and for it use by developers requiring to create a complier for languages within the scope of XBldr.

## Scope

This document is intended to be used for its development, then to be used by a developer as a reference.

## Terminology

**…** …

## Overview

The XBldr project is a complier generator application similar to Flex/Yacc. It used to create a complier by processing a grammar definition and producing source code to implement the complier. The language being implemented is abstractly known as the XLanguage.

### XLanguage

The language being implemented, from XBldr's point of view, is known as the XLanguage. When the XBldr refers to this language, terms like XLanguage or XLang are used (as a reference to an unknown entity). Part of the definition is to give the XLangauge a name to create the complier, amongst other things, with an .exe with this name. As an example, if a C# complier was to be generated, the XLanguage could be named "CSComplier", producing 'CSComplier.exe'.

An implementation of an XLangauage is XBldr itself.

### Grammar Definition

XBldr requires a Gramma Definition to generate an XLangauge complier. This Gramma Definition is of two forms: a .gdf file, and source file embedded. Generally, the gramma is initially defined as a GDF file and continued when first generated into a source file. Working on the source file is generally done when actions and their source are being developed and debugged. Modification in either form will reflect in the other if both active, though gramma development will be less efficient in the source.

* + - 1. GDF file

A .gdf file is a text file written in the XBldr language format as describe throughout this document. It defines the Gramma, Tokens, States, Actions and limited source. (Defined later).

The contents of this file are used to generate the source and are copied to the generated source as embedded Gramma Definition as comments.

This file is the desirable place to do the initial grammar development, or on going development, for the specific gramma specification. Limited gramma element actions development is done in this file, as it is more efficient to do so within the generated source.

* + - 1. Source file embedded

The gramma from a .gdf file will be copied to the source file generated as source code comments using the syntax applicable to the language concerned. The specifics for a particular language are defined later under the section for that language, including language specific features such as preprocessor constructs and line versus multi-line usage.

* + 1. **XBldr**

This is the name given to this project and collectively refers to this project as a whole. It also is the name of the application and its executable file that ...

* + - 1. **XBldrDev**

As mentioned XBldr and its Gramma Definition language ...

## Text Font Usage

Throughout this document, certain fonts are used to indicate items of a certain type or nature. 0 lists the fonts, what they mean and their usage.

|  |  |
| --- | --- |
| **Font + Settings** | **Meaning and Usage** |
| Times New Roman | Normal text –no special meaning |
| *Times New Roman + Italic* | Special proper names of various items either defined in this document or related to the subject matter of this document. |
| *Times New Roman + Italic + underline* | Various items which are soon to be defined usually in either a table, numbered or dot points or as sub-sections to the section where they appear. |
| **Times New Roman + Bold** | Acronyms and terms with specific meanings defined in section 1.3 1.3 and/or in the [ref] Glossary appendix. |
| *Arial + Italic* | Menu path to select menu or dialog box items. Each menu/tab is separated by a ‘*>*’ |
| Courier New | Code Snippets or Source, Including references to Functions, Objects, Properties, etc. |
| Tahoma | File names/paths |

Table 1 Fonts and their usage

# XBldr and XLanguage

**XBldr** and **XLanguage** (**XLang** or **XLang…**) are the names of

## XBldr

### …

## XLanguage

### …

…

# Gramma Definition

## Gramma Definition File

### …

### …

## Embedded Gramma Definition, .cs File

# Source Language Implementations

Currently, XBldr is only implemented in and for C#. This may change in the future so as XBldr can be used for other languages. New subsections will be added for these possible future versions.

## C# Implementation

### Source File embedded Gramma Definition

* + 1. Generated Source

### Inherited Parser Engine

* + - 1. Generated inheriting Parser

## …

1. Tokens

# Keywords

## Options

Options are used to initiate or modify values used in the support of a Parser or Gramma generation. …

### ?Name I!

Used to set the name of the XLanguage, and optionally as a summary. This name is used to name the assembly and .exe file. The summary is optional and will fill the '<summary>' source doc tag for the XLangClass -see '?Class'. Its usage is;

?Name: <XLangName> [Summary]

XLangName: The name of the XLanguage -must be Identifier and filename valid.

### ?Class

Used ..

### ?Style

Sets the language general style to a common format such as ‘C’ language style. This option will affect the default values of various other options such as Comments, Strings etc. A summary of the effects on various options is given below –for full details of how Style affects various options, see each individual option.

…. Summary

Available Styles are;

### ?Options ?

Used to set several options as a groupwithout the need declare them individually. The option s that can be set this way are parameterless and are indicated by a '@' symbol. Certain defaults are also available to set as option and are indicated by ths '@' symbol.

Where a Option or Default can be used, within the section that it is described, its description will include how the Option or Default is applied as a item within the ?Options list. These individual descriptions should be consulted for these details.

Its usage is;

?Options: (<Options>|<Defaults>)[',' (<Options>|<Defaults>)]\*

Options: An option to set. Option must be applicable (indcated by '@')

Defaults: A dfault to set. Defaults must be applicable (indcated by '@')

### ?Comments I!

Defines the comment indication characters and does not have a default per se. However, comments can be set by a default value for by ?Style, e.g. ?Style C.

Used to set the name of the XLanguage, and optionally as a summary. This name is used to name the assembly and .exe file. The summary is optional and will fill the '<summary>' source doc tag for the XLangClass -see '?Class'. Its usage is;

?Name: <XLangName> [Summary]

XLangName: The name of the XLanguage -must be Identifier and filename valid.

### ?Name I!

Used to set the name of the XLanguage, and optionally as a summary. This name is used to name the assembly and .exe file. The summary is optional and will fill the '<summary>' source doc tag for the XLangClass -see '?Class'. Its usage is;

?Name: <XLangName> [Summary]

XLangName: The name of the XLanguage -must be Identifier and filename valid.

### 

## Defaults

Defaults are values that can be defined for options, tokens, states and actions. They begin with a '!' symbol and define values, or control the value other elements. They are often used to set values for elements where a default value is adequate and avoids the need to explicitly define these values.

### !ASCII

### !UTF7

### !UTF8

### !UTF16

### !UTF32

### !Unicode

### !BEUnicode

Used to … Big Endian

### Token

Use to indicate that for each state Regex pattern generated, a wrapper group named <token> is generated around the Regex Pattern. This allows the Match value (token) to be extracted without knowing the group that captured it or retrieves the whole combined capture of all groups.

### !WS

Used to set the default “white space” regular expression pattern for use in token identification. See Token Identification. The default pattern is;

Figure 1 …

# Error Catching

# Appendix A –Quick Reference Guide

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol | Context | | Description |
| ` |  | `…` | Comment –note this is the alternate quote symbol |
|  |  | ` to EOL |  |
| & | GD | &name: | Token definition name header |
|  | GD, … | &name | Token reference –used within gramma |
|  |  | &Empty | Special Token reference, the “Empty” token |
| # |  | # | Line No. |
|  |  | #@ | Line position |
|  |  | […]# | Delimiter Iteration |
| … |  | tba |  |

Symbols :

` Comment -note this is

‘…’ Character Token

“…” Keyword or Literally string –interpreted by options set

=“…” Regex pattern for defining Tokens –quotes in the pattern must be escaped ‘\”’

^… State

~… Negate option, used to turn off an option that otherwise would be on (e.g. via

>…

@ Error Catch

# Line

$ Symbol Value

$name

?...: Set Option ‘…’

? “…” Conditional keyword, keyword is only defined for grammar context in which it is found

& Action Commands, e.g. &Warn

% Group and pattern reference e.g. !Symbol = (?<symbol>.\*)

! Default value/definition

%^ Precedence: No Association

%< Precedence: Left association

%> Precedence: Right association

\* Zero or More List Items

+ Concatenates tokens such that no text can be ignored between them. Used for multi-character operators etc. e.g. ‘>=’. LISTS: One or More List Items

. Defines the type for a grammar element

[] Option/List

() Explicitly defines association of grammar elements, e.g. G1 G2 | G3 G4 != G1 (G2 | G3) G4

{} Defines the action (code) of Gramma Elements

= Sets the declared grammar element to the value of this grammar element,

Or, if defining a Token Regex

<> Causes state to return to previous state

Lists and Optional Grammar:

‘[’ <GrammarList> [‘{‘ <ActionList> ‘}’] ‘]’

‘[’ <TerminalGrammarList> ‘;’ <RecursiveGrammarList> ‘]’ [ListOption]

TerminalGrammarList =

GrammarList = [Grammar {action1}; + {action2} ‘.’ {action3} ]+

= [Grammar {action1}; Grammar {action2} ‘.’ {action3}]+

= Grammar {action1} | Grammar {action2} ‘.’ {action3} GrammarList

# Appendix B