ASN2CSV
Coding Policy

Version 0.5

Revision History

Date	Version	Description	Author
2008-11-11	0.1	Initial Draft	ŽKn
2008-11-17	0.5	Rules are defined	ŽKn

Doc. No.:

1. Introduction

1.1 Purpose of this document

The purpose of this document is define rules for coding.

1.2 Document organization

The document is organized as follows:

- Section 1, *Introduction,* describes contents of this guide, used documentation during developing process etc.
- Section 2, *Rules*, defines rules

1.3 Intended Audience

The intended audience is:

- Team members

1.4 Scope

This document defines rules that all team members must follow when writing source code. This is not a mandatory document bur rather a set of guidelines. It's purpose is to allow team members to easily read and track source code written by other parties.

1.5 Definitions and acronyms

1.5.1 Definitions

Keyword	Definitions
Team members	People involved in writing source code.

1.5.2 Acronyms and abbreviations

Acronym or abbreviation	Definitions

1.6 References

2. Rules

2.1 Only one statement is allowed per line.

2.2 Do not leave unnecessary blank lines.

2.3 Use tab character. Convert spaces to tabs (most of editorsprovide this option).

2.4 Align the new line with the beginning of the expression at the same level on the previous line. i.e. Do block related statements together.

2.5 Indent tab for each block and all continuation lines have to be indented.

2.6 Understand that at any moment, user can read only 25 or so lines of code.

So, organize that code around that fact. That means (these are not strict rules):

- Have functions that are smaller than 25 lines. Best way to create new functions is to abstract some part of the problem (domain, echnical, or linguistic) and provide a function for that. It always should be possible to get such an abstraction going.
- If the function is large, break down into blocks, where each block is doing some unique activity. Use one line comment describe that activity.
- Use appropriate formatting scheme to cut down on excessive lines. For examples, ornate commenting scheme is not good. Placing empty lines that does not indicate some semantic separation to the reader is not good.
- For complex logic functions, include the algorithm before the function body and after the comment section.

2.7 Understand that code is meant to be read and understood.

That means, it should be readable, say, over the phone. That means:

- Use meaningful names. Long names are not necessarily the most meaningful. It is ok to use i and j for indexing. If there is an abbreviation, such as num, no etc. use one unique abbreviation through out the project.
- Use verbs in naming procedures, because it indicates action. Use names for functions that return values.
- Never, ever use two names that differ only in capitalization, and punctuation.

2.8 Comments should not repeat the code.

Comments are meant provide higher-level road map. That means:

- Comments should explain the domain portion, not the code.
- Comments should tell the reader why and what you are doing it, rather than how.
- In case the code is tricky, explain the how, and tell them explicitly why it is tricky.
- Use commenting style that is easy to maintain.