Style Guidelines Observed in `gtap.tab` *Release 6.2*

Robert McDougall

GTAP Research Memorandum No. 4

October 2003

Robert McDougall: Deputy Director, Center for Global Trade Analysis, Department of Agricultural Economics, Purdue University, 1145 Krannert Building, West Lafayette, IN 47907

Contents

Because `gtap.tab` is a high-profile product, and is worked on by many hands, stylistic consistency is good to have but hard to maintain.

The stylistic inconsistency of the GTAP code has followed an arch-like trajectory. Inconsistencies were rare in the original version, written almost single-handed by Tom Hertel; between versions 2 and 4 they became more common, as new matter was incorporated from other authors; since version 5 they have again become rarer, as we have ordered and regularised the code.

The style however has not ended as it began; it has evolved as we have gained experience in code maintenance.

In these notes we describe the style as of `gtap.tab` release 6.2. We make no attempt at completeness; most points are more easily absorbed by browsing the code than by conning a guide. We address especially points where the desired style has been uncertain, and on rules easily broken through inadvertency. The few small innovations introduced in release 6.2 are explained in McDougall [1].

In the code examples below, we sometimes use the string ' `|` ' to separate the code itself from our comments on it.

1 Generalities

The file comprises *comments* (sections 5–8) and *statements* (sections 9–19).

The file may also be considered as a series of *paragraphs*, blocks of non-blank lines separated by blank lines. A paragraph might contain a single statement, a series of statements, a comment, part of a comment, or both statements and comment.

The file contains both code and natural language *text*. Text appears not only in comments but also embedded in *labels* (section 14) in code statements. Text includes both *heading text* (section 2) and *plain text* (section 3). A piece of text may be either a complete sentence or series of sentences, or a *text fragment* (section 4).

Finally, the file may be considered as a hierarchy of *sections*. These include an initial "Preliminaries" section, several modules, and several appendices. These in turn are divided into smaller units: preliminaries subsections, submodules, and sub-appendices.

Some very general rules apply to the entire file:

1. **General:**

   1.1 Always use spaces never `<TAB>`s.

   1.2 Leave no trailing whitespace (easily broken).

   1.3 Use an indentation width of 4 characters.

2 Heading text

Heading text appears both in actual headings (section 6) and in outlines (section 7).

2. **Heading text:** Use headline-style capitalization:

```
RIGHT | 5-2. Demand for Imports
WRONG | 5-2. Demand for imports
```

Exception: All caps are tolerated in the initial and end-of-program headings, and in some headings in the preliminaries before the modules, for example `FILES` and `SETS` headings.

3 Plain text

Plain text is any natural language text other than heading text.

3. **Plain text:**

   3.1 Separate words with one space (easily broken).

   3.2 After a comma, leave one space (easily broken).

   3.3 After a colon, leave one space.

   3.4 After an end-of-sentence period, leave two spaces.

4 Text fragments

A *text fragment* is a piece of natural language text that is not part of a complete sentence. Most heading texts and labels and some short comments are text fragments.

4. **Text fragments:**

   4.1 Unless heading text, do not capitalize.

```
RIGHT | REG # regions in the model #
WRONG | REG # Regions in the model #

RIGHT | # Extra eq'n computes change in supply in the omitted market. #
```

   4.2 Do not leave a terminal '`.`'.

```
RIGHT | # terms trade effects decomposed by effects of different prices #;
WRONG | # terms trade effects decomposed by effects of different prices. #;

RIGHT | # Extra eq'n computes change in supply in the omitted market. #
```

5 Comments

Almost all comments in recent versions of `gtap.tab` fall into a few kinds:

- A *heading comment* (sections 6–8) contains a section heading, and perhaps other remarks.
- A *prepended comment* begins a paragraph, and typically states its contents.
- An *appended comment* follows a statement or ends a paragraph, and typically comments on its contents.

```
!<
    1-1. Virtues and Sins
    ---------------------

    [heading comment]
>!

! virtues [prepended comment] !
Set
    CARDINAL (prudence, justice, temperance, fortitude);
Set
    THEOLOGICAL (faith, hope, charity);
Set
    VIRTUE = CARDINAL union THEOLOGICAL;

Set
    DEADLY (envy, avarice, pride, gluttony, lust, wrath, sloth);
!<
    Envy always wants to come first; sloth doesn't mind going last.
    [appended comment]
>!
```

Some general rules apply to comments of all kinds:

5. **Comments:**

   5.1 Open with `!<`, close with `>!`. **Exception:** We tolerate some single-line prepended comments enclosed with just '`!`' before and after.

   5.2 Put the comment body on a separate line or lines from the opening and closing marks.

```
RIGHT |  !<
RIGHT |      Even short comment bodies need a separate line.
```

```
RIGHT |  >!

WRONG |  !< Short comment bodies need no separate line. >!
```

Exception: We tolerate some single-line prepended comments enclosed with just '`!`' before and after.

5.3 Left-justify.

5.4 Indent the opening and closing marks zero times, the comment body once:

```
!<
    The comment body begins in column 5.
>!
```

5.5 Fill lines; set the line width to 78 characters.

5.6 Leave no blank line after the opening `!<`, or before the closing `>!`.

6 Heading comments

A *heading comment* contains a *section heading*, consisting of *heading text*, an underline, and possibly an overline. It may also contain remarks pertaining to the section it heads. If these are elaborate, they may have their own *remarks headings*.

6. **Heading comments:**

6.1 Write one heading comment per section. Don't put a section heading and its first subheading in the same comment; and don't start a new comment for a remarks heading.

6.2 Leave two blank lines before a heading comment, and one after.

6.3 Mark headings with underlines and sometimes overlines:

Level 0 '`=`' overline and underline

Level 1 '`=`' overline and underline

Level 2 '`-`' overline and underline

Level 3 '`-`' underline

Level 4 '`.`' underline

6.4 Match the width of the underline or overline to the width of the heading (easily broken).

6.5 Leave two blank lines before a remarks heading.

6.6 Leave one blank line between the heading and the body of the remarks.

All this is easier to do than to describe:

```
!<
    --------------------
    1. The First Section
    --------------------
>!

Coefficient (integer)
    ONE #one#;
Formula
    ONE = 1;


!<
    ----------------------
    2. The Second Section
    ----------------------

    This is a simple remark.
>!

Coefficient (integer)
    TWO #two#;
Formula
    TWO = 2;


!<
    --------------------
    3. The Third Section
    --------------------


    Initial Remarks
    ---------------

    This is the initial remark.


    Final Remarks
    -------------

    This is the final remark.
>!
```

Amongst other kinds of remark, heading comments may include two special kinds: *outlines* (section 7) and the *version history* (section 8).

7 Outlines

Some heading comments include *outlines*, lists of subheadings in the subsequent material:

7. **Outlines:**

   7.1 Format outlines as heading text.

   7.2 Match outlines to the actual subheadings (easily broken). It should be possible to cut and paste from the outlines into the actual subheadings.

   7.3 Make each outline one level deep; don't list sub-subheadings.

8 History

The first heading comment contains the *version history*.

8. **History:** List changes in order from latest to earliest.

9 Statements

The code comprises Tablo *statements* optionally arranged into paragraphs.

9. **Statements:**

   9.1 Group statements into paragraphs at your discretion.

   9.2 Capitalize just the initial letter of the statement keyword.

```
WRONG | subset
WRONG |     TYPE is subset of CTAX;

RIGHT | Subset
RIGHT |     TYPE is subset of CTAX;

WRONG | SUBSET
WRONG |     TYPE is subset of CTAX;
```

   9.3 Most statements declare, assign to, or otherwise act on some object (set, coefficient, variable, ...). In such statements, leave a line break and an indent before the name of that object. This applies to the following statement types: `Set`, `Subset`, `Coefficient`, `Variable`, `File`, `Read`, `Write`, `Formula`, `Update`, `Display`, `Mapping`.

```
RIGHT | Set
RIGHT |     TYPE (xtax, mtax);

WRONG | Set TYPE (xtax, mtax);
```

9.4 Subject to more specific rules, leave one space between words (easily broken).

```
RIGHT | maximum size 5 read elements from file GTAPSETS header "H6";
WRONG | maximum size 5  read elements from file GTAPSETS header "H6";
```

9.5 Leave no space before the terminal ';'.

```
RIGHT | Zerodivide (nonzero_by_zero) off;
WRONG | Zerodivide (nonzero_by_zero) off ;
```

Statements contain various elements such as names (section 11), qualifiers (section 12), quantifiers (section 13), and labels (section 14), subject to their own particular rules. *Boilerplate* (section 15) is the fixed multi-word strings specified in the syntax for some statements, for example, the string `read elements from file` in `Set` statements. An *equation* (section 16) is the part of an `Equation`, `Formula`, or `Update` statement that resembles a mathematical equation. *Expressions* occur not only in equations but also in complex quantifiers and `Assertion` statements. Particular rules apply to various elements of expressions, including *fences* (section 17), *operators* (section 18), and *summations* (section 19). `Equation` statements (section 10) have some rules all their own.

10 Equation statements

Some special rules apply to `Equation` statements.

10. **Equation statements:**

    10.1 Leave a line break between the equation name and label. Do not indent the label.

    10.2 Leave a line break between the equation label and the quantifiers. Do not indent the quantifiers.

    10.3 Leave no line break before the terminal ';'.

11 Names

Users define *names* for sets, coefficients, variables, and so on. Although the Tablo language is case-insensitive, some case conventions are widely observed.

11. **Names:** File, set, equation and coefficient names are upper-case, variable names lower-case. **Exceptions:** Absolute change variables may be lower-, mixed-, or upper-case; for example, `del_taxrgc`, `CNTdpar`, `DTBAL`. A few mixed-case equation names are tolerated, for example, `TOTeq`, `c1_irEQ`. In `decomp.tab`, some coefficients are named after `gtap.tab` variables; these match in case the variables they are named after.

12 Qualifiers

Qualifiers appear immediately after the keyword in many declaratory statements.

12. **Qualifiers:** Qualifiers are all lower-case:

```
RIGHT | Coefficient (integer) (all,i,ENDW_COMM)
WRONG | Coefficient (Integer) (all,i,ENDW_COMM)
```

13 Quantifiers

Quantifiers appear immediately after the qualifiers in many declaratory statements, but after the label in `Equation` statements.

13. **Quantifiers:** Leave no spaces after the commas:

```
RIGHT | Variable (all,r,REG)
WRONG | Variable (all, r, REG)
```

14 Labels

Labels, deliminated by '`#`' characters, appear in most declaratory statements, including `Set`, `Coefficient`, and `Variable` statements.

14. **Labels:** Leave just one space inside each '`#`':

```
WRONG | SIZE_TRAD #size of TRAD_COMM set#;
RIGHT | SIZE_TRAD # size of TRAD_COMM set #;
WRONG | SIZE_TRAD # size of TRAD_COMM set  #;
```

15 Boilerplate

Boilerplate is the fixed multi-word strings specified in the syntax for some statements, for example, the string `read elements from file` in the `Set` statement.

15. **U:**se all lower case for boilerplate.

```
RIGHT | maximum size 1 read elements from file GTAPSETS header "H9";
WRONG | Maximum size 1 read elements from file GTAPSETS header "H9";
```

16 Equations

In `Equation`, `Formula`, and `Update` statements, the main content is a piece of code laid out like a mathematical *equation*.

16. **Equations:**

    16.1 Omit superfluous line breaks.

    16.2 If an equation must be broken, break it according to precedence of operators and brackets, from lower to higher: first at the '=' sign, then between brackets rather than within brackets, then between terms rather than between factors:

```
RIGHT | CNTtech_afmfdsd(m,i,r,s)
RIGHT |     = [0.01 * EVSCALFACT(s)] * VTMFSD(m,i,r,s) * atmfsd(m,i,r,s);

WRONG | CNTtech_afmfdsd(m,i,r,s) = [0.01 * EVSCALFACT(s)]
WRONG |     * VTMFSD(m,i,r,s) * atmfsd(m,i,r,s);

RIGHT | CNTtech_afr(r)
RIGHT |     = [0.01 * EVSCALFACT(r)]
RIGHT |     * sum(j,PROD_COMM, sum(i,TRAD_COMM,
RIGHT |         [VIFA(i,j,r) + VDFA(i,j,r)] * af(i,j,r)));

WRONG | CNTtech_afr(r)
WRONG |     = [0.01 * EVSCALFACT(r)]
WRONG |     * sum(j,PROD_COMM, sum(i,TRAD_COMM, [VIFA(i,j,r)
WRONG |         + VDFA(i,j,r)] * af(i,j,r)));
```

    16.3 Within a summation, break after the `sum(<index>,<RANGE>` strings and before the summand.

```
RIGHT | CNTtech_atrr(r)
RIGHT |     = [0.01 * EVSCALFACT(r)]
```

```
RIGHT |     * sum(m,MARG_COMM, sum(i,TRAD_COMM, sum(s,REG,
RIGHT |         VTMFSD(m,i,s,r) * atmfsd(m,i,s,r))));

WRONG | CNTtech_atrr(r)
WRONG |     = [0.01*EVSCALFACT(r)]
WRONG |     * sum(m,MARG_COMM, sum(i,TRAD_COMM,
WRONG |         sum(s,REG, VTMFSD(m,i,s,r)*atmfsd(m,i,s,r))));
```

16.4 Break before not after operators '=', '+', '-', '*', '/':

```
RIGHT | CNTtech_amsr(r)
RIGHT |     = [0.01 * EVSCALFACT(r)]
RIGHT |     * sum(i,TRAD_COMM, sum(s,REG, VIMS(i,s,r) * ams(i,s,r)));

WRONG | CNTtech_amsr(r)
WRONG |     = [0.01 * EVSCALFACT(r)] *
WRONG |     sum(i,TRAD_COMM, sum(s,REG, VIMS(i,s,r) * ams(i,s,r)));
```

16.5 Indent equations.

16.6 If you break an equation at the '=', indent the RHS once more than the LHS:

```
RIGHT |   qp(i,r) - pop(r)
RIGHT |       = sum{k,TRAD_COMM,
RIGHT |           EP(i,k,r)*pp(k,r)} + EY(i,r)*[yp(r) - pop(r)];

WRONG |   qp(i,r) - pop(r)
WRONG | = sum{k,TRAD_COMM, EP(i,k,r)*pp(k,r)} + EY(i,r)*[yp(r) - pop(r)];
```

17 Fences

Fences are braces, brackets, and parentheses, used to control precedence in mathematical expressions.

17. **Fences:**

17.1 In equations, group terms with brackets not parentheses or braces:

```
RIGHT | qgd(i,s) = qg(i,s) + ESUBD(i) * [pg(i,s) - pgd(i,s)];
WRONG | qgd(i,s) = qg(i,s) + ESUBD(i) * (pg(i,s) - pgd(i,s));
WRONG | qgd(i,s) = qg(i,s) + ESUBD(i) * {pg(i,s) - pgd(i,s)};
```

17.2 Omit unnecessary fences. **Exception:** In a long sum or product (three or more terms or factors), terms or factors may be grouped together for expressiveness.

```
RIGHT |   CNTtech_aor(r)
RIGHT |       = [0.01 * EVSCALFACT(r)]
RIGHT |       * sum(i,PROD_COMM, VOA(i,r) * ao(i,r));

WRONG |   CNTtech_aor(r)
WRONG |       = [0.01 * EVSCALFACT(r)]
WRONG |       * [sum(i,PROD_COMM, VOA(i,r) * ao(i,r))];
```

17.3 Leave no space inside the brackets:

```
RIGHT | pcgdswld = sum(r,REG, [NETINV(r) / GLOBINV] * pcgds(r));
WRONG | pcgdswld = sum(r,REG, [ NETINV(r) / GLOBINV ] * pcgds(r));
```

18 Operators

18. **Operators:**

18.1 Leave a space before and after each binary operator '=', '+', '-', '*', '/':

```
RIGHT | UELASPRIV(r) = sum{i,TRAD_COMM, CONSHR(i,r) * INCPAR(i,r)};
WRONG | UELASPRIV(r) = sum{i,TRAD_COMM, CONSHR(i,r)*INCPAR(i,r)};
```

Exception: Leave no space in `orig_level` qualifiers:

```
RIGHT | Variable (orig_level=1.0)(all,i,NSAV_COMM)(all,r,REG)
WRONG | Variable (orig_level = 1.0)(all,i,NSAV_COMM)(all,r,REG)
```

18.2 Leave no space after unary '+' or '-'.

```
RIGHT | TOT(i,r,"pimport","percent") = -c3_ir(i,r);
WRONG | TOT(i,r,"pimport","percent") = - c3_ir(i,r);
```

19 Summations

The *summation* construct

`sum(<index>,<range>, <expression>)`

occurs frequently in the code.

19. **Summations:**

19.1 Fence with parentheses not brackets or braces:

```
RIGHT | UELASPRIV(r) = sum(i,TRAD_COMM, CONSHR(i,r) * INCPAR(i,r));
WRONG | UELASPRIV(r) = sum[i,TRAD_COMM, CONSHR(i,r) * INCPAR(i,r)];
WRONG | UELASPRIV(r) = sum{i,TRAD_COMM, CONSHR(i,r) * INCPAR(i,r)};
```

19.2 Leave no space inside the parentheses:

```
RIGHT | qtm(m)
RIGHT |     = sum(i,TRAD_COMM, sum(r,REG, sum(s,REG,
RIGHT |       VTMUSESHR(m,i,r,s) * qtmfsd(m,i,r,s))));

WRONG | qtm(m)
WRONG |     = sum(i,TRAD_COMM, sum(r,REG, sum(s,REG,
WRONG |       VTMUSESHR(m,i,r,s) * qtmfsd(m,i,r,s) )));
```

19.3 Leave no space after the first comma (after the dummy index):

```
RIGHT | CONSHR(i,r) = VPA(i,r) / sum(k,TRAD_COMM, VPA(k,r));
WRONG | CONSHR(i,r) = VPA(i,r) / sum(k, TRAD_COMM, VPA(k,r));
```

19.4 Leave one space after the second comma (after the range):

```
RIGHT | TIM(r) = sum(i,TRAD_COMM, sum(s,REG, MTAX(i,s,r)));
WRONG | TIM(r) = sum(i,TRAD_COMM,sum(s,REG, MTAX(i,s,r)));
```

19.5 In multiple summations, match the summation order to the index order in the summand:

```
RIGHT | VTMD(m,s) = sum(i,TRAD_COMM, sum(r,REG, VTMFSD(m,i,r,s)));
WRONG | VTMD(m,s) = sum(r,REG, sum(i,TRAD_COMM, VTMFSD(m,i,r,s)));
```

Exception: Where the derivation of the formula or equation leads naturally to some particular order of summation, use that order; for example, in the equations for the EV contributions of the technical change variables `afe`, `ava`, and `af`, the industry index `j` naturally precedes the input index `i`.

References

[1] R.A. McDougall. *`gtap.tab` Release 6.2*. GTAP Research Memorandum No. 3. October 2003.