Designing BibTeX Styles
BibTeX スタイルの作り方* ,†

Oren Patashnik(訳：松井正一)

January 31, 1988(翻訳版：1991 年 1 月 1 日)

5　参考文献スタイルをハックする

本稿は，実際には BibTeX の一般的なドキュメントである “BibTeXing” [?] の最終節であり，第 5 節からはじまる (そしてこの節だけで終る)．しかし先のドキュメントは BibTeX の利用者向けであり，本稿はスタイル作成者向けであるので，物理的に分離したものとなっている．とはいえ，この文書の読者は “BibTeXing” に精通している必要があり，本稿で参照している節，節番号は 2 つの文書が 1 つのものと仮定している．

本節と標準スタイルファイルのドキュメントである `btxbst.doc` で[1] 既存のスタイルファイルの修正方法，新たなスタイルファイルの作成方法を説明する．もしちゃんとしたスタイルファイルを作ろうと思っているのであれば，スタイルに関して van Leunen [?] に，タイプセットに関して Lamport [?] と Knuth [?] に，さらに場合によっては，細かな互換性の問題から *Scribe* [?] に精通していなければならない．せっせと仕事をする合間に，Strunk and White による名小冊子 [?] を読むことを薦めるが，読まないのであれば，少なくてもデータベース中のエントリと参考文献リストを見て，BibTeX が複数の名前をどう扱っているかを調べること[2]．

新たなスタイルを作る場合には，望みのものに近い既存のスタイルを修正するのがよい[3]．これは古いスタイルを BibTeX version 0.99 用に修正する場

---

*翻訳の部分は原著者の許可を得て配布するものであり，翻訳について，jBibTeX に関する記述については松井に問い合わせられたい．

†jBibTeX(version 0.30)[?] の説明も含む．

[1] 訳注：日本語対応版は `jbtxbst.doc` である．

[2] 訳注：日本語対応版での漢字コード著者名の扱い方についても調べること．

[3] 訳注：新たなスタイルを作る時だけでなく，既存のスタイルを修正する時でも，直接 `.bst` ファイルを修正するのではなく `btxbst.doc`，`jbtxbst.doc` を修正する方が，再利用可能

合にもあてはまる．(私自身が 4 つの非標準スタイルを修正した経験からの忠告である)．古い (version 0.98i) スタイルのために作った関数を新しいスタイルファイルに入れる時には，代入関数 (:=) の引数の順番が変更されたことに注意しないといけない．スタイルファイルが完成したら，XAMPL.BIB データベースの全てを使って[4]，標準エントリの全ての型に対して望みの結果になることを確認すればよい．

標準スタイルの虫，やりたいことができない場合には Oren Patashnik に文句を言うこと[5]．

5.1 概要

参考文献スタイルはスタックマシンに対する命令を後置演算子形式で記述する．標準スタイルのドキュメントを見れば，どう書くかを知るのは難しいことではないが，本稿ではもっと詳しく説明する (希望があればさらに詳しくする)．

基本的にはスタイルファイルは，名前のない言語で，プログラムを記述することにより，どのようにしてデータベースのエントリから参考文献リストを作るかを BibTeX に教えるものである (日本語対応スタイルでは jBibTeX に教えることである．以下 BibTeX に当てはまることは jBibTeX にも当てはまるので，いちいちことわらない)．(以下では「エントリ」は「エントリ・リスト」のことであり，文脈から明らかな場合には単に「リスト」とする)．このプログラム言語には，次の細節 (subsection) で説明する 10 個のコマンドがある．これらのコマンドは言語のオブジェクト：定数，変数，関数，スタック，エントリ・リストを操作するものである．(注意：この文書中の用語は説明を簡単にするために選んだものであり，BibTeX 本来の用語とは若干異なる．例えばこの文書中の「変数」,「関数」はどちらも BibTeX にとっては関数である．BibTeX のエラーメッセージを解釈する時にはこのことを念頭におくこと)．

BibTeX の関数には**組み込み**のもの (第 5.3 節で説明する) と MACRO か FUNCTION で定義するものの 2 種類がある．

スタイル作成で最も時間のかかる作業は FUNCTION コマンドを使って関数を作成する，あるいは既存のものを変更することである．(実際には，前述の

性が高くなる，運が良ければ 1 つの修正で複数のスタイルを修正できる，などの利点がある．

[4] 訳注：\nocite{*} で行える．

[5] 訳注：日本語用のものについては松井正一 (matsui@denken.or.jp) に知らせること．

参考文献に精通するのに要する時間の方が長いが，それが終った後の話しである)．

例として関数の一部を取り出したものを見てみよう．文字列型変数 label，整数型変数 lab.width があり，label の後ろにに ‘a’ を連結し，lab.width を 1 増やしたいとしよう．

```
. . .
label "a" * 'label :=          % label := label * "a"
lab.width #1 + 'lab.width :=   % lab.width := lab.width + 1
. . .
```

1 行目で先ず label はその値をスタックにプッシュする．次に "a" が文字列定数 ‘a’ をスタックに置く．続いて組み込み関数 * がスタックトップの 2 つの文字列をポップし，それらを連結したものをプッシュする．'label はその変数名をプッシュする．最後に組み込み関数 := が変数名と連結結果をポップし，代入を実行する．BibTeX はスタイルファイル中の % 以降はコメントとして扱う．2 行目も同様であるが，プッシュする整数定数として ‘#’ と ‘1’ の間にスペースのない #1 を使っている．

スペースの数はいくつでもよく，スペース，タブ，改行はいくつあっても 1 つとみなされる (後で簡単に説明するように，コマンドの途中では改行しないほうがよい)．

文字列定数としては，ダブルクォートで囲った中には印字可能文字なら何を書いてもよい[6]．文字列定数に限って BibTeX は大文字と小文字の区別をする．さらに文字列定数の中のスペースの数は意味がある．また文字列定数は行をまたがってはならない (すなわち文字列の始めと終りのダブルクォートは同じ行になければならない)．

変数，関数の名前は数字で始まってはならず，LaTeX book の 143 ページにある 10 個の禁止文字を含んではならないが，それ以外の印字可能文字なら何を使ってもよい[7]．さらに (ASCII コードの) 大文字と小文字の区別はしない．

変数，定数の型としては整数型と文字列型しかない (論理値は整数の 0(偽) と 1(真) として実現している)．変数の種類としては 3 種類ある．

大域変数 INTEGERS あるいは STRINGS コマンドで宣言された整数型あるいは文字列型の変数．

[6] 訳注：日本語対応版では漢字コードでもよい．
[7] 訳注：日本語対応版では漢字コードでもよい．日本語プログラミングできる:-)．

エントリ変数 ENTRY コマンドで宣言された整数型あるいは文字列型の変数．エントリリストのエントリ毎に別々の値を持つ (例えば `label` にエントリのラベルを持たせることができる)．

フィールド 読みだし専用の文字列型の変数であり，データベースの情報が格納される．値は READ コマンドでデータベースが読み込まれることで設定される．エントリ変数同様に各々のエントリ毎に別々の値を持つ．

5.2 コマンド

10 個のコマンドがある．ENTRY, FUNCTION, INTEGERS, MACRO, STRINGS の 5 個は変数の宣言，関数の定義のためであり，READ はデータベースの情報を読み込み，EXECUTE, ITERATE, REVERSE, SORT) の 4 個はエントリを操作し，出力を作り出す．すべてのコマンドを大文字で示したが，BibTeX はコマンド名の大文字と小文字の区別をしない．

いくつかの制限:ENTRY と READ の数は 1 つでなければらない．ENTRY コマンド, すべての MACRO コマンド, FUNCTION コマンドのいくつか (次節 `call.type$` 参照) は READ コマンドの前になければならない．また READ コマンドはエントリ操作，出力の 4 つのコマンドより前になければならない．

(本質的ではないが) コマンドの間には 1 つ以上のブランク行を置き，関数定義の中にはブランク行を置かないようにするとよい．これによって BibTeX の構文エラーからの回復を助けることができる．

コマンドの引数は中括弧 (`{ }`) に入れる．本節で述べられていない構文に関する事項は標準ファイルのドキュメント[8] を参照されたい．以下で 10 個のコマンドの説明を行う．

ENTRY フィールドとエントリ変数を宣言する．3 引数であり，各々の引数は (空かもしれないが) 変数名のリストである．それらはフィールド，整数型のエントリ変数，文字列型のエントリ変数である．また BibTeX が相互参照で使うために自動的に宣言する `crossref` フィールドと，SORT コマンドで使うソートキーとして `sort.key$` が，文字列型のエントリ変数として自動的に宣言される．以上の変数はエントリ・リストのエントリ毎に別々の値を持つ．

EXECUTE 関数を 1 つ実行する．引数は関数名 1 つである．

[8] 訳注： `btxbst.doc`, `jbtxbst.doc` のこと．

FUNCTION 新たな関数を定義する．2 引数であり，最初が名前で 2 番目が定義である．関数は使う前に定義しなければならない．したがって再帰関数は許されない．

INTEGERS 整数型の大域変数を宣言する．引数は変数名のリスト 1 つである．文字列変数の長さを制限する大域的変数として entry.max$ と global.max$ が自動的に宣言される．このコマンドはスタイルファイル中にいくつあってもよいが，変数は使う前に宣言しなければならない．

ITERATE 現在のリストの順番にリストのエントリに対して 1 つの関数を実行する (最初はリストは引用順に並んでいるが，SORT によって順番は変わる)．引数は関数名 1 つである．

MACRO 文字列マクロを定義する．引数は 2 つで，最初が変数名，関数名と同じように扱われるマクロ名で，次がダブルクォートで囲まれた文字列であり，その定義である．月名の 3 文字の省略形とよく使われる論文誌の省略形を定義しておくこと．この定義は利用者のデータベース中の定義で置き換えることができるから，利用者に変更してもらいたくないものは，同じ構文の FUNCTION を使って定義すればよい．

READ エントリ・リスト中のエントリのフィールドの値をデータベースから探索して設定する．引数はない．データベース中のエントリに，対応するフィールドの値がない場合には (必ずしもデータベース中にすべてのフィールドが書かれているわけではない) そのフィールドの値には欠測値のマークが付く．

REVERSE ITERATE コマンドとまったく同じことをエントリリストの逆順に実行する．

SORT 文字列型のエントリ変数 sort.key$ を使ってエントリリストをソートする．引数はない．

STRINGS 文字列型の大域変数を宣言する．引数は変数名のリスト 1 つである．このコマンドはスタイルファイル中にいくつあってもよいが，変数は使う前に宣言しなければならない．

5.3 組み込み関数

組み込み関数の説明の前に，組み込みオブジェクトついて少し説明しておく．組み込みのエントリ文字列変数として `sort.key$` があり，ソートを行うスタイルでは値を設定しなければならない．組み込みのフィールドとして，第 4 節で述べた相互参照機能のための `crossref` がある．さらに整数型の大域変数 `entry.max$` と `global.max$` があり，その値は BibTeX の内部定数に設定されている．BibTeX から警告メッセージが出ないよう，文字列への代入の前に，代入する文字列長をこれらの値より短くしておかねばならない．

現在組み込み関数は 37 個ある[9]．すべての組み込み関数の名前は最後が '$' で終る．以下では「1 番目」,「2 番目」というのはスタックからポップされた順番を意味する.「リテラル」はスタックの要素であり，整数値，文字列，変数名，関数名，あるいは欠測フィールドを表す特別の値のいずれかである．ポップされたリテラルの型が誤っていると BibTeX は文句をいった後，関数の返す値の型に対応して，整数値の 0 あるいは空文字列をプッシュする．

`>` 2 つの (整数) リテラルをポップし，比較を行い，2 番目が 1 番目より大きければ 1 をそうでなければ 0 をプッシュする．

`<` 上と同様 (小さければ).

`=` スタックトップの (ともに整数あるいは文字列の) 2 つのリテラルをポップして比較し，等しければ 1 を，そうでなければ 0 をプッシュする．

`+` (整数の)2 つのリテラルをポップしその和をプッシュする．

`-` (整数の)2 つのリテラルをポップしその差をプッシュする．(2 番目から 1 番目を引く).

`*` 2 つの (文字列) リテラルをポップし，それらを連結したものをプッシュする (逆順，2 番目の後に 1 番目を連結する).

`:=` 2 つのリテラルをポップし 1 番目 (大域変数かエントリ変数) に 2 番目のリテラルの値を代入する．

`add.period$` スタックトップの (文字列) リテラルをポップし，'}' でない最後の文字が '.', '?', '!' のいずれでもなければ '.' を最後に加えた結果をプッシュする．いずれかであれば元の文字列をプッシュする．

JBibTeX での注意：全角の '．', '。', '?', '!' の後にも '.' は加えない．

[9] 訳注：日本語対応版では，組み込み関数を 1 つ追加しているので，38 個ある．

`call.type$` その名前がエントリのエントリ型の名前である関数を実行する．例えば，エントリの型が `book` であれば `book` という関数が実行される．`ITERATE` コマンドの引数として指定された場合には，`call.type$` はエントリの情報を `.bbl` ファイルに出力する．不明なエントリ型 (未定義の型) に対しては `default.type` を実行する．したがって各々の標準エントリ型に対応するもの以外に `default.type` という型に対応する関数を `READ` コマンドの前に定義しておかねばならない．

`change.case$` 2 つの (文字列) リテラルをポップし，1 番目のリテラルの指定にしたがって，以下に説明する形式で，2 番目の文字列の大文字/小文字変換を行う (注意：次文で「文字」とは中括弧のレベル 0，最も外側の中括弧のレベルの文字のことであり，第 4 節で説明した「特殊文字」を除くそれ以外の文字は変換されない)．1 番目のリテラルが '`t`' であれば，文字列の 1 番始めの文字およびコロンの後にスペースが 1 個以上続く文字列の最初の文字はそのままで，それ以外を小文字に変換する．1 番目のリテラルが '`l`' であれば，すべての文字を小文字に変換する．1 番目のリテラルが '`u`' であれば，すべての文字を大文字に変換する．そして変換結果の文字列をプッシュする．2 つのリテラルの型が文字列型でない場合には文句を言い，空文字列をプッシュする．リテラルの型は合っているが，変換指定が上記の何れでもない場合には，文句を付けた後，単に 2 番目の文字列をそのままの形でプッシュする (もう 1 つの注意：変換指定文字の大文字/小文字は区別されない．すなわち `t` と `T` は同じものとされる)．

`chr.to.int$` スタックトップの (文字列) リテラルをポップし，その長さが 1 文字であることを確認した後，その ASCII コードの整数をプッシュする．

JBIBTEX での注意：漢字コード文字に対応するために，リテラルが漢字コード文字の場合には 2 バイトを 1 文字と数え，値としては 1 バイト目のコード値 (EUC コード値) をプッシュする．ただしこれは JBIBTEX version 0.10 との互換性のために残したものであり，仕様変更される可能性が高い．

`cite$` エントリに対応する `\cite` コマンドの引数の文字列をプッシュする．

`duplicate$` スタックトップのリテラルをポップしそのコピーを 2 つプッシュする．

`empty$` スタックトップのリテラルをポップし，それが欠測フィールドであるか，あるいは文字列中に空白文字以外が無い場合に，整数の 1 をプッシュし，それ以外の場合には 0 をプッシュする．

`format.name$` スタックトップの 3 つのリテラル (順に文字列，整数，文字列) をポップする．最後の文字列リテラルが名前リスト (要素が個人名に対応) であり，整数リテラルがリストの何番目を取り出すのかの指定であり，最初の文字列リテラルが名前をどの様にフォーマットするのか (次節で説明する) の指定である．フォーマットされた名前の文字列をプッシュする．

JBibTeX での注意: この関数では全角のスペースは半角のスペースと同じとみなす (より正確にいうと全角のスペースは半角のスペースに変換されて処理される)．また，全角の句点 “，” と “、” も “␣and␣” と同様に扱うので，漢字コード氏名は句点で区切って並べることができる．

`if$` 3 つのリテラル (順に 2 つの関数リテラル，整数リテラル) をポップし，最後のリテラル値の整数値が 0 より大きい時には 2 番目のリテラルを実行し，そうでなければ 1 番目のリテラルを実行する．

`int.to.chr$` スタックトップの (整数) リテラルをポップし，それを ASCII コードとして，対応する文字を 1 文字の文字列に変換したものをプッシュする．

`int.to.str$` スタックトップの (整数) リテラルをポップし，それを (一意に定まる) 文字列に変換してプッシュする．

`missing$` スタックトップをポップし，それが欠測フィールドであれば整数の 1 を，そうでなければ 0 をプッシュする．

`newline$` 出力バッファに蓄積されている情報を `bbl` ファイルに書き出す．出力バッファが空である場合に限り空行を書き出す．`write$` は適当に改行を行なうから，空行を書き出す時，あるいは明示的に改行したい時だけに，この関数を使うのがよい．

`num.names$` スタックトップの (文字列) リテラルをポップし文字列中の名前の数，すなわち，大文字/小文字を無視して，中括弧のレベル 0 の位置にあり，前後が空白文字である “and” 部分文字列の数に 1 を加えた値，をプッシュする．

JBibTeX での注意：この関数では全角のスペースは半角のスペースと同じとみなす (より正確にいうと全角のスペースは半角のスペースに変換されて処理される)．また，全角の句点 “，” と “、” も “␣and␣” と同様に扱うので，漢字コード氏名は句点で区切って並べることができる．

`pop$` スタックトップをポップするがプリントはしない．不必要なリテラルを取り除くのに使う．

`preamble$` データベースファイルから読み込んだ`@PREAMBLE` すべてを連結した文字列をプッシュする．

`purify$` スタックトップの (文字列) リテラルをポップし，空白文字，ハイフン ‘-’ とタイ ‘~’ (これらはスペースに変換される) のいずれでもないアルファベット以外の文字,「特殊文字」に付随した制御文字列 (コマンド; control sequence) に含まれるアルファベット文字を除去した文字列をプッシュする[10] ．

`quote$` ダブルクォート文字 (‘"’)1 文字からなる文字列をプッシュする．

`skip$` 何もしない．

`stack$` スタックの内容を全部ポップしてプリントする．スタイル作成時のデバッグ用．

`substring$` スタックトップの 3 つのリテラル (順に長さ *len*，開始位置 *start* の 2 つの整数リテラル，文字列リテラル) をポップする．*start* が正ならば文字列の先頭から数えて，(最初が 1 文字目として)*start* 文字目からの，*start* が負ならば文字列の終りから数えて (最後の文字が 1 文字目として)，$-start$ 文字目からの，連続する (高々)*len* 文字の部分文字列をプッシュする．

JBibTeX での注意：この関数は 1 文字は 1 バイトからなるものとして処理するので，漢字コード文字列では指定した文字数の半分の数の漢字コードが得られる．また漢字コードの 1 文字の間で切られないように開始位置，文字数 (バイト数) の調整を行なう．調整は開始位置が漢字コードの 1 バイト目に一致していれば．開始位置はそのまま，2 バイト目であれば 1 バイト目から取り出すように調整する．終了位置が漢字コードの 1 バイト目であれば 2 バイト目まで取り出すように調整す

[10] 訳注：漢字コード文字列なら変化しない．

る．したがって指定した長さより最大2バイト長い文字列となる．これは日本語用でないスタイルファイルを使った場合でもそれなりの出力が得られるようにするためである．

`swap$` スタックトップの 2 つのリテラルの順序を入れ替える．

`text.length$` スタックトップの (文字列) リテラルをポップし，アクセント付き文字 (正確には第 4 節で定義されている「特殊文字」) は，(対応する右中括弧が欠けていても)，1 文字として数えた時の文字数をプッシュする．

JBibTeX での注意：漢字コード 1 文字は 2 文字と数える．

`text.prefix$` 2 つのリテラル (順に整数リテラル *len*，文字列リテラル) をポップし，文字列の先頭から (高々)*len* 文字の連続した文字列をプッシュする．`substring$` に似ているが，この関数では，「特殊文字」は対応する右中括弧が欠けていても，複数の ASCII 文字から構成されていても，1 文字として数える．

JBibTeX での注意：この関数では特殊文字以外の文字は，1 文字は 1 バイトからなるものとして処理するので，漢字コード文字列では指定した文字数の半分の数の漢字からなる文字列が得られる．また漢字コードの 1 文字の間で切られないように，終了位置が漢字コードの 1 バイト目であれば 2 バイト目まで取り出すように調整する．したがって指定した長さより最大 1 バイト長い文字列となる．これは日本語用でないスタイルファイルを使った場合でもそれなりの出力が得られるようにするためである (例えば `alpha` ではラベルを著者名の姓の先頭 3 文字を用いて作成するが，漢字コードの中間で終らないように 2 文字にしてしまうと情報量が少なくなる，また漢字コードは特殊文字と考えて処理するとラベルが長くなり過ぎるのでこのような仕様とした)．

`top$` スタックトップをポップしその内容を端末とログファイル `blg` にプリントする．デバッグに役立つ．

`type$` 現在のエントリの型 (`book`，`article` など) の文字列をプッシュする．知らない型あるいは未定義の型であれば空文字列をプッシュする．

`warning$` スタックトップの (文字列型) リテラルをポップし，警告メッセージであることを示してプリントする．また，出力した警告メッセージの数を 1 増やす．

`while$` 2 つの (関数) リテラルをポップし，最初の関数を実行した結果であるスタックトップの値が 0 より大きい間，2 番目の関数を繰返し実行する．

`width$` スタックトップの (文字列型) リテラルをポップし，その印字幅をある相対単位に基づいて計算した整数値をプッシュする (現在は 1987 年 6 月バージョンの *cmr*10 フォントに基づいて，1/100 ポイントを 1 として計算する[11])．この関数は文字列を文字通りに扱う，すなわち，(対応する右中括弧が欠けていても) 「特殊文字」は特別に扱う以外は，たとえある文字が TEX で特別な意味を持っていても，文字はそのままの形でプリントされるものとして幅を計算する．ラベル文字列の幅を比較するためのものである．

`write$` スタックトップの (文字列) リテラルをポップし，出力バッファに書き出す (その結果，バッファが一杯になれば `bbl` ファイルにも書き出される)．

`is.kanji.str$` JBIBTEX のみで定義されている組込み関数であり，スタックトップの (文字列) リテラルをポップし，文字列中に日本語文字 (全角文字) が含まれていれば (整数の)1 を，そうでなければ 0 をプッシュする．

組込み関数 `while$` と `if$` ではスタック上に 2 つの関数リテラルが必要である．これらは関数名の前にシングルクォートを付けるか，`FUNCTION` コマンドを使って関数を定義したくなければ，その定義を (`FUNCTION` コマンドの 2 番目の引数を中括弧で囲んで) そのまま書けばよい．例えば次に示す関数の一部分は，`label` が空文字列でなければ，その最後に文字 'a' を付け加える．

```
. . .
label "" =
  'skip$
  { label "a" * 'label := }
if$
. . .
```

名前をクォートする関数は，`skip$` のような組込み関数である必要はなく，例えば，フィールドでも，自分で定義した関数でもよい．

[11] 訳注：漢字コードの幅情報は正確でないので注意が必要．

5.4 名前のフォーマット

名前には何が入っているのか? これについては第 4 節で少し詳しく説明した．名前は「名 (first)」パート, von パート, 「姓 (last)」パートと Jr パートからなり，それぞれは名前トークンのリストからなり，それぞれのパートは空かもしれないが，名前が空でなければ姓パートは必ず空ではない．この小節では組込み関数の `format.name$` に指定するフォーマット指定文字を説明する．

例として，非常に長い名前を考えてみよう．データベースのエントリ [?] に次のフィールドがあるとしよう．

```
author = "Charles Louis Xavier Joseph de la Vall{\'e}e Poussin"
```

また「姓 カンマ イニシャル」(“last name comma initials”) の順で名前をフォーマットしたいとしよう．次のようなフォーマット指定文字列を使うと，

```
"{vv~}{ll}{, jj}{, f}?"
```

BibTeX はフォーマットされた文字列として次を作り出す．

```
de~la Vall{\'e}e~Poussin, C.~L. X.~J?
```

この例をもっと詳しく見てみよう．このフォーマット指定文字列には，名前のそれぞれの構成要素に対応する，中括弧レベルが 1 の 4 つの**断片** (*pieces*) がある．対応する構成要素がなければ (この名前では Jr 要素)，その断片の中のすべてが無視される．中括弧のレベル 0 にあるものは (この例では，たぶんタイプミスであろうが，‘?’ の文字は) そのままの形で出力されるが，この機能はあまり使わないであろう．

フォーマット指定の部分中では二重文字 (double letter) は構成要素のすべてを使うことを，単文字 (single letter) は省略形にしたものを使うことを BibTeX に指定する．これらの指定文字は中括弧のレベル 1 になければならず，それ以外の文字は (後で説明してある文字以外のほとんどすべての文字は) そのままの形で使われる．von パートの最後にあるタイ (~) は任意タイであり，BibTeX が必要と判断した場合にのみ出力され，そうでなければスペースとして出力される．本当にタイを出力したいのであれば 2 つ書いておけば，つまり`{vv~~}`としておけば (1 つだけ) 出力される．断片の最後の文字であるタイは任意タイとされるが，それ以外は通常の文字として扱われる．

BibTeX は名前トークンの**間に**デフォールトの区切り文字列を書き出す．これはスペースかタイのどちらか適した方，あるいは省略形トークンではピリ

オドの後にスペースかタイの適した方を付けたものである．しかしフォーマット指定の最後のトークンの後にはデフォールト区切り文字列を書き出さないから，先の例で言えば‘J’の後にはピリオドは付かない．したがって次のように指定して，

```
"{vv~}{ll}{, jj}{, f.}"
```

前の出力例の疑問符をピリオドに置き換えたフォーマット結果を得ることができる．名前に「名」パートがない場合に対応するように，ピリオドは疑問符の位置でなく「名」断片の中に置かねばならないことに注意しよう．

BibTeX のデフォールトの区切り文字を置き換えたいのなら，明示的に指定する必要がある．例えば von パートと姓パートのすべてのトークンの先頭文字を，間にスペースを置かずに連結したラベルを作りたいのであれば，次のように指定しなければならない．

```
"{v{}}{l{}}"
```

こうすればフォーマット結果の文字列として‘dlVP’が得られる．置き換えたいすべての部分毎に指定が必要であり (この例では両方に空文字列が指定されている)，指定は単文字，二重文字のすぐ後になければならない．フォーマット指定文字列の中括弧のレベル 1 の中では，以上で述べた文字以外を書くことはできない．

JBibTeX での注意点：

- 漢字コード表記された名前では，姓と名の間にスペース (半角でも全角でも) を入れておけば，‘f’は Family name(姓) に，‘l’は Last-token(名) に対応することになるので，通常の表記順に名前が書ける．しかしスタイルの作成では注意が必要である．これは format.name$ を完全な日本語対応版としていないからであるが，実際に日本語対応のスタイルファイルを作成した経験からいうと，漢字コード表記された名前とそれ以外の名前に対して，同じフォーマット指定で format.name$ を呼び出すことは殆どないので，あえてこのままにしてある．しかし混乱の種かもしれない．

- jabbrv で姓のみを出力できるようになどの目的で，JBibTeX ver. 0.20 以降では姓と名の間にスペース (半角でも全角でも) を置くことを標準としている．しかしスペースがない場合にも結果がおかしくならないように，フォーマット指定には注意が必要である．例えば日本語スタイルの中では次のようなコーディングを行なっている．

```
editor is.kanji.str$                                % 漢字コードの編集者?
    {editor #1 "{ff}" format.name$ duplicate$  % 姓を取り出す
       empty$                                       % 空?(姓と名の間に空白がない)
          {pop$ editor #1 "{ll}" format.name$}  % Last token を取り出す
          'skip$                                    % noop
       if$
    }
    {editor #1 "{vv~}{ll}" format.name$}        % 英文編集者
if$
```

5.5 JBibTeX の標準スタイル

JBibTeX の標準スタイルとしては plain，alpha，abbrv，unsrt に対応して jplain，jalpha,jabbrv，junsrt が作成されている．さらに情報処理学会論文誌 tipsj，情報処理学会欧文論文誌 jipsj，電子情報通信学会論文誌 tieic，日本オペレーションズリサーチ学会論文誌 jorsj，人工知能学会誌 jsai，ソフトウェア科学会誌 jssst 用のスタイルも作成されている．これらのスタイルで行なっている日本語対応の主な変更は以下の通りである．詳しくは jbtxbst.doc を参照されたい．

1. 著者名が日本語かどうかを is.kanji.str$ を使って判定し，名前のフォーマットの方法を変える．
2. 著作名に日本語が含まれる場合には強調指定を付けない．
3. ページ範囲指定を Pages から pp．に変更した．
4. yomi フィールドがあれば，その情報をソーティングキーを作る時に著者名，編集者名の代りに使う．