many text-processing and desktop-publishing systems still lack *APL* support, it remains difficult to achieve high printing quality in publications composed of text and *APL* code. Several extensions to existing text processors have been implemented (cf. [Hohti, Kanerva 88]). However, most of them support either only the symbols of one *APL* dialect or only one machine or operating system platform. [Hohti, Kanerva 88] already demonstrated the usefulness of TEX for *APL* typesetting. The authors produced a METAFONT description for *APL* primitive symbols and a set of TEX macros to support Digital's *APL* interpreter for the VAX-11 series.

In this paper we present our solution to the problem: An *APL* publishing system consisting of an *APL* front end and a LATEX document style option. The *APL* front end automatically converts *APL* material into LATEX code which you can `\input` into any standard LATEX document. The LATEX document style option `apl.sty` provides macros defining all *APL* characters as combinations of standard LATEX symbols, thus relieving us from the burden of designing new fonts and the user from the task of incorporating them into the LATEX system. As additional benefit, size and type style of the *APL* symbols can be changed by the familiar LATEX commands (e.g. `\Large`, `\sf`).

Compared with the approach of Hohti and Kanerva mentioned above, our solution offers the following advantages:

- Several *APL* dialects are supported (currently *APL2*, Dyalog *APL*, I-*APL*, Sharp *APL*, and *APL*.68000).
- No additional fonts are required.
- *APL* symbols for all LATEX sizes and type styles are available.
- We provide support for automatic typesetting of *APL* objects.

There are some disadvantages, however; they are higher TEX interpretation overhead and higher TEX memory usage.

2 The *APL* Publishing System

The *APL* publishing system consists of two parts, the *APL* front end and the LATEX document style option `apl.sty` which communicate via a carefully designed interface of TEX macros (see Figure 1). This ensures that both parts of the system can be modified independently.

Each of these modules is composed of two layers (see Table 1). The main task of the *low level formatting* layer is the printing of single *APL* symbols. The *APL* front end maps each symbol into a TEX macro name and produces files to be `\input` into LATEX documents. The LATEX style option `apl.sty` contains one macro definition for each *APL* character.

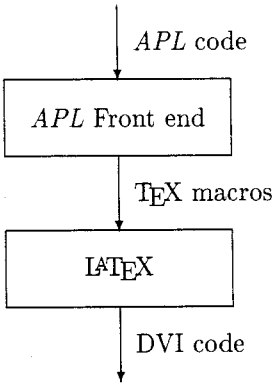


Figure 1: Modules of the Publishing System

APL code is more than just a stream of *APL* symbols. The *high level formatting* layer knows about functions, operators, arrays, and expressions. Our *APL* front end provides special functions for typesetting these objects. The LATEX style option defines the corresponding environments.

	APL front end	LATEX document style option
Low level formatting	symbol translation	symbol construction
High level formatting	*APL* language elements	logical document elements

Table 1: Layers of the Publishing System

3 Typesetting *APL* Symbols

If using a few *APL* symbols in an ordinary document is all you need, you can forget about the *APL* front end. Simply adding the option `apl` to your preferred LATEX document style (e.g. `\documentstyle[12pt,apl]{article}`) enables you to state in your paper, e.g.:

> By combining the simple *APL* symbols ○ and ⋆ we obtain the compound symbol ⊛.

The code to produce this statement is:

```
\begin{quotation}
By combining the simple \APL\ symbols
\APLcircle\ and \APLstar\ we obtain the
compound symbol \APLcirclestar.
\end{quotation}
```

In fact, you can typeset all simple and compound symbols of *APL2*, as we have defined macros for all of them. Tables 2 and 3, respectively, show them together with their macro names.

\APLalpha	α	\APLnotgreater	$\leq$
\APLbar	$-$	\APLnotless	$\geq$
\APLcircle	$\bigcirc$	\APLomega	ω
\APLcolon	:	\APLoverbar	$\bar{}$
\APLcomma	,	\APLplus	+
\APLdel	∇	\APLquad	$\square$
\APLdelta	$\triangle$	\APLquery	?
\APLdieresis	¨	\APLquote	'
\APLdivide	$\div$	\APLrho	ρ
\APLdot	.	\APLrightarrow	$\rightarrow$
\APLdownarrow	$\downarrow$	\APLrightbracket	]
\APLdowncaret	$\vee$	\APLrightparen	)
\APLdownshoe	$\cup$	\APLrightshoe	$\supset$
\APLdownstile	$\lfloor$	\APLsemicolon	;
\APLdowntack	$\perp$	\APLslash	/
\APLepsilon	ϵ	\APLslope	\
\APLequal	$=$	\APLstar	$\star$
\APLgreater	$>$	\APLstile	$\mid$
\APLiota	$\imath$	\APLtilde	$\sim$
\APLjot	$\circ$	\APLtimes	$\times$
\APLleftarrow	$\leftarrow$	\APLunderbar	_
\APLleftbracket	[	\APLuparrow	$\uparrow$
\APLleftparen	(	\APLupcaret	$\wedge$
\APLleftshoe	$\subset$	\APLupshoe	$\cap$
\APLless	$<$	\APLupstile	$\lceil$
\APLmathbar	$-$	\APLuptack	$\top$
\APLnotequal	$\neq$		

Table 2: Simple *APL2* Symbols

IBM was the first company to implement *APL* but it did not remain the only one. Companies such as I. P. Sharp and Dyadic Systems have produced their own versions of the language. These and other companies, however, introduced only a few symbols not found in *APL2*. We have added twenty additional symbols to the *APL2* character set to support typesetting Dyalog *APL*, I-*APL*, Sharp *APL*, and *APL*.68000 (see Table 4).

As you have probably guessed from the names in Tables 2, 3, and 4 we stick to a naming convention in order to minimize name clashes with other macro packages and also help users remembering the macro

\APLcirclebar	$\ominus$
\APLcircleslope	$\oslash$
\APLcirclestar	$\circledast$
\APLcirclestile	$\odot$
\APLdelstile	∇
\APLdeltastile	$\triangle$
\APLdeltaunderbar	$\underline{\triangle}$
\APLdeltilde	$\underline{\nabla}$
\APLdieresisdot	$\ddot{\cdot}$
\APLdowncarettilde	$\not\vee$
\APLdowntackjot	$\perp\!\!\!\circ$
\APLdowntackuptack	$\perp\!\!\!\top$
\APLepsilonunderbar	$\underline{\epsilon}$
\APLequalunderbar	$\underline{=}$
\APLiotaunderbar	$\underline{\imath}$
\APLleftbracketrightbracket	$\|$
\APLquaddivide	$\boxdot$
\APLquadjot	$\boxcirc$
\APLquadquote	$\boxed{'}$
\APLquadslope	$\boxed{\backslash}$
\APLquotedot	!
\APLslashbar	$\not/$
\APLslopebar	$\not\backslash$
\APLupcarettilde	$\not\wedge$
\APLupshoejot	$\cap\!\!\!\circ$
\APLuptackjot	$\top\!\!\!\circ$

Table 3: Compound *APL2* Symbols

names. All macro names start with the \APL prefix, followed by the name of the symbol used in the *APL* literature. The symbol names for *APL2* characters are taken from [IBM 85]. For those characters (cf. Table 4) which are not included in the IBM list we have invented consistent names. We always use symbol names, not the name of *APL* functions these symbols might represent. The name of a compound *APL* symbol is the concatenation of the names of the simple *APL* symbols it is created from.

As can be seen in Figure 2 which shows the character set (the atomic vector $\square$AV) of *APL2*, not all *APL* characters are fancy symbols, and the language uses ordinary alphanumeric characters as well. To allow for a clean interface between the *APL* front end and the LaTeX part of our system, we decided to define macros for these characters as well. Their names are constructed as follows:

- Each macro name starts with \APL.
- For each letter we append the upper or lowercase letter, if the letter is underlined we prefix the letter with "u".
  Capital letters: \APLA, ..., \APLZ.

\APLOUT	⍵
\APLasciipound	£
\APLbarcomma	⍪
\APLcaret	^
\APLdiamond	◇
\APLdieresiscircle	⍜
\APLdieresisdel	⍫
\APLdieresisjot	⍤
\APLdieresisstar	⍣
\APLdieresistilde	⍨
\APLdieresisuptack	⍦
\APLlefttack	⊢
\APLnotequalunderbar	≠
\APLquaddownarrow	⍗
\APLquadleftarrow	⍇
\APLquadrightarrow	⍈
\APLquaduparrow	⍐
\APLrighttack	⊣
\APLstilebar	⍭
\APLtheta	θ

Table 4: Symbols Used in APL Dialects

Lowercase letters: \APLa, ..., \APLz.
Underlined capital letters: \APLuA, ..., \APLuZ.
Underlined lowercase letters: \APLua, ..., \APLuz.

- For numbers we simply append their names: \APLzero, ..., \APLnine.

The tiny numbers in the atomic vector of Figure 2 correspond to positions for which no printable characters are defined by *APL2*. In case the *APL* front end encounters a nonprintable character, e.g. the one at position 20 in ⎕AV, it generates \APLmiss{20}. The definition of the LaTeX macro \APLmiss determines the printed representation of this character (the default macro in our style just prints the corresponding number in style \tiny).

Let us close this section with one more example of typesetting *APL* symbols:

```
\APLquaddivide\APLA\ corresponds to
$A^{-1}$ in mathematical notation
and \APLcircleslope\APLA\ corresponds
to $A^{T}$.
```

displays as:

⍤A corresponds to A^{-1} in mathematical notation and ⍉A corresponds to A^T.

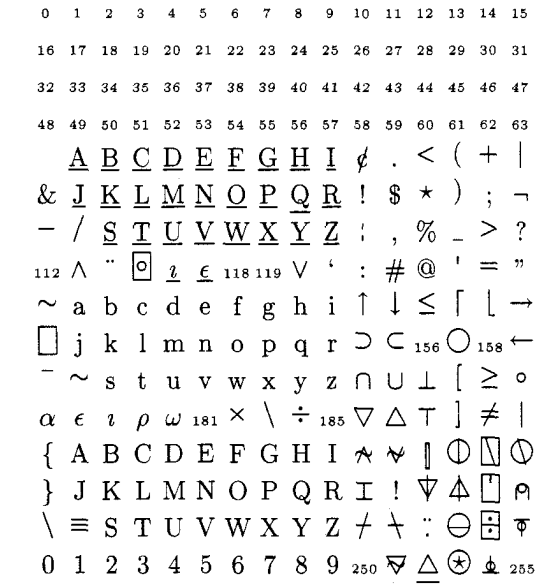


Figure 2: The Atomic Vector of *APL2*

4 Typesetting *APL* Objects

Typing the name of an occasional *APL* symbol within a normal text is not a real nuisance to the author of *APL* texts. But typesetting a larger piece of *APL* code certainly is. Imagine a function named △TREE, which implements a recursive tree traversal algorithm:

```
    ∇Z←CLASS_LIST △TREE ROOT;
      DEPTH;LIST;I;RECLIST;SUPERCLASS
[1]   Z←,⊂ROOT
[2]   →(∨/(,⊂ROOT)≡¨CLASS_LIST)/
      CYCLIC
[3]   I←2 ⎕TF 2⊃GET_MEM '△CLASS'(
      ROOT,'.SUPERCLASS')
[4]   →((0⍴⊂1⍴' ')≡SUPERCLASS)/0
[5]   I←(⊂CLASS_LIST,⊂ROOT)△TREE¨
      SUPERCLASS
[6]   Z←((Z⍳Z)=⍳⍴Z)/Z←Z,↑,/,¨,I
[7]   →0
[8]   CYCLIC:Z←0⍴0
    ∇
```

In order to print just the beginning of the header of the function, you would have to type:

```
\APLspace\APLspace\APLspace\APLspace%
\APLspace\APLdel\APLZ%
\APLleftarrow\APLC\APLL\APLA\APLS\APLS%
\protect\APLunderbar\APLL%
\APLI\APLS\APLT\APLspace\protect\APLdelta%
```

```
\APLT\APLR\APLE\APLE%
\APLbr\APLspace\APLR\APLO\APLO\APLT\APLbr%
```

Obtaining the familiar function layout used in *APL* textbooks would require additional code. What is more, besides being awkward the whole process is error-prone: Almost certainly it will result in a printout different from the *APL* code.

Therefore we strongly recommend automatic translation of *APL* code. We provide an *APL* front end which transforms *APL* objects into logical document elements which can be `\input` into LaTeX documents. This guarantees consonance between the original *APL* code and its listing and is also more convenient.

For all *APL* language elements we have defined *APL* functions and corresponding LaTeX environments. Our system supports the typesetting of:

- an array displayed by the interpreter,
- an array in boxed representation,
- a function or operator displayed by the built in *APL* del-editor (∇-editor),
- a function or operator displayed by *APL*'s canonical representation function ⎕CR,
- a direct definition of a function or operator,
- an *APL* expression input by the user.

Apart from minor modifications we have used traditional layout conventions for all language elements. For example, the convention that user input is six spaces indented can be traced back to the very first implementation of *APL*. Another traditional convention states that if a line of *APL* code does not fit on a single line of the display, the rest of the code is wrapped around and continues on the next line. In some functions this rule may lead to line breaks in the middle of names. Since *APL* identifiers can be up to 255 characters long line breaking within names cannot be avoided in general.

As you can see, arrays and functions can be typeset in various ways. For example, the above listing of the *APL* function ΔTREE was printed by the following *APL* expression:

'TREE' PRTEX_FN 'ΔTREE'

The *APL* function PRTEX_FN produces the file `tree.tex` as output. The *APL* front end not only maps each character into the corresponding TeX macro but it also produces the line numbers in brackets and the surrounding LaTeX environments in order to guarantee uniform display of functions throughout the document.

In the following we present examples for each of the cases mentioned above. At the same time, the examples give us the opportunity to demonstrate variations of type style and size.

4.1 Typesetting *APL* Arrays

The interpreter usually displays arrays as text matrices on the screen. For example, the matrix X is displayed as:

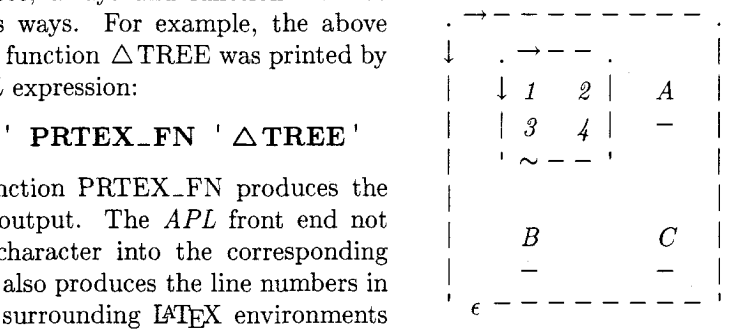
```
1  2      A
3  4

B         C
```

The above printout is typeset by the following code which is automatically produced by the *APL* front end:

```
% AR1 X
\begin{APLarray}
\APLmb{\APLspace}\APLmb{\APLone}
\APLmb{\APLspace}\APLmb{\APLtwo}
\APLmb{\APLspace}\APLmb{\APLspace}
\APLmb{\APLspace}\APLmb{\APLA}
\APLspace\par
  ...
\end{APLarray}
```

Note that the structure of X has been preserved by automatically enforcing fixed spacing. A closer examination of the code reveals that we have simulated fixed spacing by boxing each character of the array (`\APLmb` does this).

Experienced *APL* programmers recognize the structure of X at the first glance: X is a two by two matrix whose upper left element is a two by two matrix. However, since the use of nested arrays is typical for second generation *APL*s like *APL2* and Dyalog *APL*, another representation of arrays exists which shows the structure in a more explicit manner:



Most of the work for typesetting the boxed representation of X shown above is done by the *APL*

function DISPLAY which usually comes with the *APL* system (e.g. [IBM 85]). Our *APL* front end just translates the characters generated by this function; the same LaTeX environment is used for both array representations. We only sketch the code for the boxed representation:

```
{\it
\begin{APLarray}
\APLmb{\APLdot}\APLmb{\APLrightarrow}
\APLmb{\APLbar}\APLmb{\APLbar}
  ...
\end{APLarray}
}
```

In order to demonstrate the ease of changing type styles we have decided to put the generated code unit into an *italics* environment. This is the reason for all letters and numbers in the boxed representation being in *italics*. Otherwise, they would have been roman.

4.2 Typesetting *APL* Functions

The following *APL* function is printed in del-editor style:

```
     ∇Z←MEAN X
[1]    Z←(+/X)÷⍴X
     ∇
```

A larger font has been selected by inserting the generated code into a \Large environment:

```
\begin{Large}
% FNS MEAN
\begin{APLfns}
\begin{APLfnsline}{}{\APLdel}
\APLZ\APLleftarrow\APLM\APLE\APLA\APLN
\APLspace\APLX
\end{APLfnsline}
\begin{APLfnsline}{\APLleftbracket\APLone
\APLrightbracket}{}
  ...
\end{APLfns}
\end{Large}
```

The canonical representation of an *APL* function is simply a text matrix. Since older *APL* systems only provide arrays of uniform datatype and rectangular shape, padding of short lines with spaces is performed. In contrast to the del-editor style, the canonical representation is typeset like an *APL* array with fixed spacing and without line numbering. For the canonical representation of MEAN a small typewriter type style was chosen:

```
Z←MEAN X
Z←(+/X)÷⍴X
```

You notice immediately that we have used the LaTeX commands \small and \tt to produce this effect:

```
{\small\tt
% CR MEAN
\begin{APLcr}
\APLmb{\APLZ}\APLmb{\APLleftarrow}
\APLmb{\APLM}\APLmb{\APLE}
\APLmb{\APLA}\APLmb{\APLN}
\APLmb{\APLspace}\APLmb{\APLX}
\APLspace\par
  ...
\end{APLcr}
}
```

In addition to the del-editor representation and the canonical representation of an *APL* function we provide means for formatting direct definitions of functions, which are supported by only a few *APL* dialects, e.g. I-*APL*:

```
fib: z,+/¯2↑z←fib ⍵−1 : ⍵=1 : 1
```

This direct definition of a function computing Fibonacci numbers is due to [Iverson 87] and has been formatted as follows:

```
\begin{APLline}
\APLf\APLi\APLb\APLcolon\APLspace\APLz
\APLcomma\APLplus\APLslash
\APLoverbar\APLtwo\APLuparrow\APLz
  ...
\end{APLline}
```

Note, that the APLline environment allows ligatures within names.

4.3 Typesetting *APL* Expressions

Finally, our system enables the user to typeset *APL* expressions. The expression

```
X←2 2⍴(2 2⍴⍳4) 'A' 'B' 'C'
```

which happens to be the one used to generate the matrix X (our example for formatting arrays) is printed by:

```
% EXPR
\begin{APLbold}\begin{APLexpr}
\APLX\APLleftarrow\APLtwo\APLspace
```

```
\APLtwo\APLrho
\APLleftparen\APLtwo\APLspace\APLtwo
  ...
\end{APLexpr}\end{APLbold}
```

The environment `APLexpr` provides the traditional six space indentation for user input.

4.4 Typesetting User Dialogues and Workspaces

The above examples demonstrate the usefulness of the six basic document elements we provide. Besides being useful on their own, we can combine them to form higher level units. In the current version of the *APL* front end, support for typesetting a dialogue and an *APL* workspace listing is included.

A dialogue is a pair of user input and interpreter response. We typeset the user input as an *APL* expression and the interpreter response as an *APL* array. The dialogue

```
      (1(2 3)4
SYNTAX ERROR
          (1(2 3)4
          ∧
```

was generated by entering the following *APL* expression

```
'E' PRTEX_DIALOG '(1(2 3)4'
```

Hopefully, most of the dialogues will not result in a syntax error as the above one. But automatically typesetting examples with errors is very convenient for describing *APL*'s error trapping mechanisms in a text book.

An example for the printout of an *APL* workspace would be too space-consuming to be included here. It basically is implemented by combining the printout of arrays, functions and operators intertwined with LaTeX sectioning commands. LaTeX's table of contents considerably increases the utility of a workspace listing.

5 Implementation Details

For symbol construction three internal macros had to be defined. The first, `\@APLmath`, puts a math symbol into a boxed math environment and adjusts spacing. The second, `\@APLmraise`, puts a math symbol into a raised and boxed math environment and adjusts spacing. The third, `\@APLovly`, simulates backspacing and overstriking on a typewriter by overlaying two boxes. The quad symbol □ required special construction in order to generate readable compound symbols such as ⍁, ⍇, ⍈. Full reconstruction was needed only for the *APL* symbol ⍞ which is a quad symbol with a short vertical rule. The first version of this symbol used a single quote instead of the vertical rule and looked rather awkward.

One disadvantage of our solution is high TeX memory consumption. We have used `\let` commands wherever possible in order to cut down memory usage. Typesetting *APL* symbols for this text has cost us approximately 6,800 words of TeX memory. Typesetting the workspace of the *APL* front end (32 functions, 11 variables, 23 pages) has cost a total of 72,800 words of TeX memory with 29,000 words used for the *APL* symbols. We recommend TeX with 262,141 words of memory.

APL lines are sometimes too long to be printed in a single line. When displaying them on the screen, this problem is usually resolved by wrapping them to the next lines without adding any hyphen. Thus, line breaks can occur anywhere in an *APL* line, in the middle of *APL* expressions or even in names. As we use one macro for typesetting each *APL* character, the normal TeX hyphenation algorithm no longer works.

In the definition of each *APL* symbol which cannot be used in an identifier the macro `\APLgb` is used, which allows breaks with a penalty of -10. To achieve line breaks in the emergency case the preprocessor inserts the macro `\APLbr` into names longer than 15 characters at regular intervals.

When typesetting *APL* arrays (cf. Figure 2), a fixed spaced font is necessary to preserve its shape. We imitate fixed spaced fonts by simply putting a box of fixed width around all characters.

For typesetting bold *APL* code the special environment `APLbold` is defined. It sets `\bf` and `\boldmath` and adjusts the thickness of rules used in symbol construction.

6 Conclusion

In this paper we have presented our solution to the *APL* typesetting problem: An *APL* publishing system consisting of a LaTeX document style option and an *APL* front end. No additional fonts are needed. We have given short examples which demonstrate the usefulness of this approach. The system has already been used by the authors to prepare several

articles published in *APL* Quote Quad, the journal of the *APL* user community.

METAFONT could be used to improve the printing quality of some symbols (cf. [Hohti, Kanerva 88]). However, it would be necessary to create a whole *APL* font family (different sizes and type styles) to obtain the flexibility of our system. We could incorporate special *APL* fonts without any change in the *APL* front end as soon as they become available.

The *APL* front end is currently implemented for *APL2* and Dyalog *APL* and can be obtained from the authors. Further porting is intended. The LaTeX document style option will be submitted to the Clarkson and Aston archives as well as to the German TeX server at Heidelberg.

The authors would appreciate comments and suggestions for the improvement of the style as well as comments with regard to *APL* symbols not available in this style.

References

[Camacho et al. 87] Camacho A., Chapman P., Ziemann D. (1987), *I-APL Instruction Manual for PC Clones*, I-APL Limited, St. Albans, Herts, England.

[Dyadic Systems Ltd. 85] Dyadic Systems Limited (1985), *Lynwood Dyalog APL User Guide*, Dyadic Systems Limited, Farnborough, Hampshire, England.

[Falkoff, Iverson 73] Falkoff A. D., Iverson K. E. (1973) "The Design of APL", *IBM Journal of Research and Development*, V17, N4, reprinted in: Falkoff A. D., Iverson K. E. (1981), *A Source Book in APL*, APL Press, Palo Alto.

[Hohti, Kanerva 88] Hohti A., Kanerva O. (1988), "Typesetting APL with TeX", *APL Quote Quad*, V18, N3, p13–16.

[IBM 85] IBM Corporation (1985), *APL2 Programming: Language Reference*, IBM Corporation, San Jose, California.

[I.P. Sharp 85] I.P. Sharp Associates Limited (1985), *SHARP APL/PC Handbook*, I.P. Sharp Associates Limited, Toronto, Canada.

[Iverson 63] Iverson K. E. (1963), "Formalism in Programming Languages", ACM Working Conference on Mechanical Language Structures, Princeton N.J., reprinted in: Falkoff A. D., Iverson K. E. (1981), *A Source Book in APL*, APL Press, Palo Alto.

[Iverson 87] Iverson K. E. (1987), "A Dictionary of APL", *APL Quote Quad*, V18, N1.

[Knuth 86] Knuth D. E. (1986), *The TeXbook*, Computers & Typesetting A, Addison-Wesley, Reading, Massachusetts.

[Lamport 86] Lamport L. (1986), *LaTeX: A Document Preparation System*, Addison-Wesley, Reading, Massachusetts.

[Micro APL Ltd. 86] Micro APL Ltd. (1986), *APL.68000 for the Apple Macintosh*, Micro APL Ltd., London, England.

[The University of Chicago Press 82] University of Chicago Press (1982), *The Chicago Manual of Style* 13^{th} *Edition*, The University of Chicago Press, Chicago, USA.

◇ Andreas Geyer-Schulz
Department of Applied Computer Science
Vienna University of Economics and Business Administration
Augasse 2–6
A-1090 Vienna, AUSTRIA
ANDREAS@AWIWUW11.BITNET

◇ Josef Matulka
Department of Applied Computer Science
Vienna University of Economics and Business Administration
Augasse 2–6
A-1090 Vienna, AUSTRIA
MATULKA@AWIWUW11.BITNET

◇ Gustaf Neumann
Department of Management Information Systems
Vienna University of Economics and Business Administration
Augasse 2–6
A-1090 Vienna, AUSTRIA
NEUMANN@AWIWUW11.BITNET