A Variation on "Take"

Frederick Macaskill
LPA Software, Inc.
290 Woodcliff Drive
Fairport, NY 14450
Tel: 716-248-9600 (Fax: 716-248-9100)

The dyadic mixed primitive function Take can be used to selectively take data out of an array or to increase the size of the dimensions of an array. With the exception of a scalar, Take cannot be used to increase the rank of the array.

When "over taking" with Take the array is filled out with zeroes if it is a numeric array and blanks if it is a character array. The left argument is the size of each dimension to take, the, right argument is the variable which is to be taken.

When over taking a mixed array we run into difficulty. Should the system be filling out with zeroes or blanks? If so why? Is there any logical reason why we should be filling out with zeroes and blanks? The assumption is that zero and blank are the typical representatives of the classes of data called numeric and character. But this may not be the case.

View data as being alphabet soup. The letters are floating in a background "substance." When we take our data from the mixture we will take with the data the background in which the data lies. APL assumes this to be blanks. This is not necessarily so. Similarly, numeric data may be taken from an environment where the background could he anything but zero.

To solve this problem it is proposed that a system variable, ⎕*FILL*, be introduced. This would be a two-element vector where the first element is the numeric fill item and the second element the character fill item. By default these would be zero and blank.

From the above:

```
      ⎕FILL← ¯32768 '*'
      A← 1 2 3
      5↑A
1 2 3 ¯32768 ¯32768
      B← 'ABC'
      5↑B
ABC**
```

Similarly, Expand also pads out data with a fill character:

```
      1 0 1 0 1 0 \ B
A*B*C*
```

This enhancement to the language would make Expand and Take more general in nature. ■

GDDM Emulator for Dyalog APL for Windows

Andrei Kondrashev
Lingo Allegro U.S.A., Inc.
203 N. LaSalle Street, #2100
Chicago, IL 60601 USA
E-mail: 71303.3224@CompuServe.com

This article presents the experience of developing a graphics auxiliary processor for Dyalog APL/W in the MS Windows environment using Dynamic Data Exchange (DDE) technology. The auxiliary processor implements a subset of the IBM Graphical Data Display Manager (GDDM) that supports the operation of IBM APL2 GRAPHPAK under Dyalog APL/W.

Dyalog APL for Microsoft Windows by Dyadic Systems Ltd. [1] is a well-implemented true Windows application. APL/W gives an APL programmer a British dialect of APL2 combined with a powerful and convenient environment. However, many important Windows functions have not been implemented in the first release, in use at Lingo Allegro. For example, graphics is not supported. It will have to be included in future versions. As we had graphical packages which we would like to run under Dyalog APL/W, we had to find some way to write our own graphics extension for APL/W.

Dyalog APL/W offers two ways to write extensions. The first way is auxiliary processors which are built into the APL system and appear in the workspace as external defined functions. The second is auxiliary processors which are connected to the APL system via shared variables. The shared variables technique is implemented in Dyalog APL/W with the help of the DDE protocol, which is a standard mechanism for interprocess communication in Microsoft Windows. We have chosen the second method, because this is a universal approach that permits us to run our extensions with APL/W and other Windows programs that support DDE sessions. Additionally, the DDE approach doesn't require special libraries and a compiler; we could use the standard Microsoft Software Development Kit (SDK). As we found out later on, this approach also requires less memory.

The DDE protocol is based on the client-server model and is a set of messages and rules that specify the method of transferring data from one application to another through common global memory blocks. There are three possible kinds of link between a client and a server: cold, warm, and hot. These types differ by the actions of a server when its data are changed.

When we made a decision regarding the kind of graphics we would like to have, we chose the