To: Distribution

From: Lindsey Leroy Spratt

Date: 12/07/81

Subject: Toward a unification of data manipulation on Multics.

1 ABSTRACT

This document proposes the unification of common data manipulations. The heart of this unification is a Unified Data Representation, the vector and vector array. Several subroutines are proposed which operate on this Unified Data Representation. There are also commands proposed which apply these subroutines to data stored in a new kind of database. Some of the services provided by the subroutines to both "user" and "system" applications are:

- displaying,
- editing, and
- sorting data.

Some interface extensions are proposed for MRDS and lister subroutines. Also, it is proposed that metering commands be extended to place their output in the new-style database. It is also proposed that the implementation of certain portions of LINUS may be done (or re-done) to take advantage of these new interfaces and facilities.

Prototypes of the commands and subroutines exist. The implementation of the commands and subroutines which constitute the new facilities should require from three to six person-months.

Many thanks to Matt Pierret for his extensive aid in the development of the prototype command and subroutine utilities.

There is a continuum meeting for discussing the material presented in this MTB at:
>udd>Multics>Spratt>mtg>vector_util_, short name "vu".

Comments should be sent to the author:

via Multics Mail:
Spratt.Multics on either MIT Multics or System M.

via US Mail:
Lindsey Spratt
Honeywell Information Systems, inc.
575 Tech Square
Cambridge, Massachusetts 02139

via telephone:
(HVN) 261-9321, or
(617) 492-9321

CONTENTS

		Page
1	Abstract	i
2	Introduction	1
3	Intended System Uses	2
4	Other potential system uses	3
5	The vector	3
6	The vector array	3
7	Implementation issues	4
8	The prototype.	4
9	Overview of the utilities.	4
	9.1 Subroutines for creation . . .	4
	9.2 Subroutines for manipulating .	5
	9.3 Commands for the vector database	5

2 INTRODUCTION

This is a proposal for a broadly applicable technology for coping with tabular data. This technology is broadly applicable in two distinct ways: First, many services are proposed for manipulating tables of data. Second, there are many instances where these services can be used since tables of data are ubiquitous on Multics. Many different facilities on Multics produce output which is, for instance, displayed in some variation on a table. The proposed technology allows the centralization of the functions of sorting, displaying, editing and evaluating any data which can be represented as a table.

The proposal has two components. The most important component is the concept of a Unified Data Representation. The Unified Data Representation is the standard "plug" by which different pieces of software can be joined, much as the RS232 connector helps many different pieces of hardware to communicate. For the Unified Data Representation to be successful, it must satisfy a couple of constraints: it must be a sufficiently general technique that widely varied systems of software can use it; and, data represented in this fashion must be self-descriptive, capable of cogent interpretation by a program without additional description.

The second component of the proposal is a set of utilities to work with this Unified Data Representation. These utilities are commands and subroutines which support the creation, editing, and display of the Unified Data Representation.

The basic model for the Unified Data Representation is the table of data. Rather than connect the Unified Data Representation terminologically with established data management concepts, none of which has exactly the right connotations, a new terminology has been adopted; the dimension, vector and vector array. This nomenclature is used to describe a simple table of data in Table 1.

In the example table displayed in Table 1, a row is analogous to a vector, a column is analogous to a dimension, and the entire table is analogous to a vector array.

```
  heroes     region      favorite color|  <-- Dimension Names
per capita                             |
                                       |
    .03      Kansas      mauve         |  <-- Vector 1
    1        Canada      puce          |  <-- Vector 2
-------------------------------------+
     ^           ^           ^
     |           |           |
Dimension   Dimension   Dimension
    1           2           3
```

Table 1. An example table labeled as a Vector Array.

The subroutine utilities for dealing with vectors are in vector_util_. These utilities are for use by portions of the Multics operating system and by general user applications. The only major requirement for them to be useable in a particular application (system or user) is for the application to be manipulating tabular data. If this is the case, the data can be cast (if it is not already) in the Unified Data Representation and it is convenient to communicate this data to the utilities, and to other applications which use the Unified Data Representation.

There is a command interface to a simple database manager which is heavily oriented toward manipulations implemented via these subroutine utilities. This database manager allows for insertion of vectors, retrieval and display of vectors, deletion of vectors, and editing (adding dimensions, changing the values of dimensions, deleting dimensions) vectors. This database manager provides a convenient way to store and manipulate data in the Unified Data Representation format. The subroutine utilities provided for manipulating data in the Unified Data Representation are useful/useable even if one does not choose to use the vector database manager, the vector database manager is an(other) application of the vector subroutine utilities.

3 INTENDED SYSTEM USES

There are (so far) three areas of the system which I propose be extended to speak in the Unified Data Representation tongue.
1) MRDS - add to dsl_ operations to: store the data described by a vector_array into a specified relation and databse; retrieve data from a database and format the retrieved data as a vector_array; and, delete data described by a vector_array from a specified relation. Each vector can be considered to correspond to either a portion or all of a tuple.
2) lister - add to the lister subroutine interface operations to:

store the data described by a vector_array into a lister database; retrieve data from a lister database formatted as a vector_array; and, delete records described by a vector_array from a lister database. Each vector is considered to correspond to a lister record.
3) metering commands - extend selected metering commands to place their output in vector databases. Extend metering subroutines to return output as vector arrays. The vectors should have in them, in addition to the data currently collected, the name of the "meter" and the sequence_number of the invocation of the meter. This sequence number should be extracted from (and updated in) the vector database in which the metering information is to be stored.

4 OTHER POTENTIAL SYSTEM USES

Portions of the implementation of LINUS can be altered to use the Unified Data Representation and some of the attendant utilities.

5 THE VECTOR

The most basic idea for implementing a set of general data management subroutine utilities is a general expression for data. This is the "vector" structure, which has two forms, the typed vector and the print vector. In both forms, the vector is an array of "dimensions". Each dimension has a name and a value. In the print vector, the values are all varying character strings, ready to be displayed or edited. In the typed vector, the values are in any of a wide variety of data type formats, ready for sorting or insertion into a database.

6 THE VECTOR ARRAY

Vectors don't appear in isolation, however. They are always part of a vector array. The vector array structure has a "dimension_table" which defines all of the dimensions present in any of the vectors in the array, and an array of pointers to vector structures. The definition of a dimension includes the name of the dimension and its data type description. For the typed_vector_array, the data type description consists of a data type number and a "size", interpreted according to the data type. For the print_vector_array, the definition has (in addition) the maximum length value for all values in that dimension in the vectors associated with the array. This information is useful when displaying the vectors.

7 IMPLEMENTATION ISSUES

There exists a set of functional proto-types, the primary limitation of which is the restriction to the print representation of data (character strings). To implement much of what is proposed here-in requires "only" the conversion of these proto-types to whatever form the final design assumes. The implementation of this proposal's utilities is in the range of 3 to 6 person-months of effort.

8 THE PROTOTYPE.

A prototype version of the command interface outlined below exists in >udd>Multics>Spratt>vd.ssd>bound_object>bound_vector_db_. There is a series of info files in the directory >udd>Multics>Spratt>vd.ssd>info.

9 OVERVIEW OF THE UTILITIES.

As noted above there are both subroutine and command utilities. The subroutine utilities fall into two categories: creation of the vector array and vectors; and, manipulation of the vector array.

9.1 Subroutines for creation

init_typed_array
: initializes a typed_vector_array (creating an "empty" array) with a given set of dimension definitions.

general_add
: adds a typed_vector to a typed_vector_array. The typed vector is given by a series of dimension names and values, from which this utility builds a typed_vector structure.

add
: adds a typed_vector to a typed_vector_array, the typed_vector being specified by providing a value for each of the dimensions defined in the typed_vector_array to which it is being added. The values must be given in the calling sequence in the same order as they appeared in the init_typed_array call that was used to create/initialize the array.

cv_typed_to_print
: converts a typed_vector_array to an equivalent print_vector_array, converting all of the associated

vectors from typed_vector structures to print_vector structures.

cv_print_to_typed
converts a print_vector_array to an equivalent typed_vector_array, converting the associated vectors from print_vector structures to typed_vector structures.

9.2 Subroutines for manipulating

sort
sorts a typed_vector_array according to specifications provided by the caller.

display
displays a print_vector_array.

edit
invokes an interactive subsystem for editing a print_vector_array. It must be provided an input_array, and can optionally be provided with entry points to invoke to delete the input_array from a data base and to insert the result_array (constructed as a result of the editing) into a data base. The subsystem can be invoked non-interactively as well.

evaluate
evaluates a vector using selected dimensions of the vector and an expression into which the dimension values are to be substituted. The substitution is done using the "do" active function. The result of the substitution is evaluated by invoking it as an active string (using cu_$af). The result of this invocation is returned as the evaluation of the vector.

external_evaluate
evaluates a set of vectors. A value is extracted from each vector of the set, then this list of values is provided as arguments to be substituted into the external expression. The substitution is done using active function "do". The substituted form of the expression is evaluated as an active string. The value extracted from the vector, mentioned above, may be specified by identifying a single dimension from which the value is to be extracted; or, it can be specified by doing an "evaluate" of a provided expression and list of dimensions to be used as arguments to the expressions.

9.3 Commands for the vector database

vector_insert
: inserts a vector into a vector database. A vector database is created the first time there is an attempt to insert a vector in it.

vector_delete
: deletes a vector from a vector database.

vector_display
: displays a selected set of vectors from a vector database. This command uses both the sort and display subroutine utilities mentioned above.

vector_edit
: edits one or more vectors in a selected set of vectors extracted from a vector database. The editing can include any mixture of: deleting dimensions, adding dimension, changing the values of dimensions. If desired by the user, this command invokes an editing subsystem request loop. This command uses the edit subroutine utility, which uses the disply, evaluate, and external_evaluate subroutine utilities, all mentioned above.