Distr.
RESTRICTED

LC/DEM/R.252
Series A, N°304
29 December 1995

ORIGINAL: ENGLISH

C E L A D E
Latin American Demographic Centre

A NEW OPEN STRUCTURE AND LANGUAGE FOR THE winR+ STATISTICAL PROCESSOR

(Working Paper)

TABLE OF CONTENTS

BACKGROUND

Present situation

Language
Experience has proven that the Redatam language is confusing for users. It looks like a procedural language: FOREACH, WRITE and an object oriented language: DEFINE, FREQUENCY, AVERAGE etc... The FOREACH command is difficult to understand and use without any surprises! There are no external functions.

Database content
The statistical processor is limited to the data originally created by the system itself (BIN files). Any data must be imported into the Redatam database prior to its use.

Database Variables
Only "stored on disk" variables are available. The concept of "Computed" variables is not available.

Output

- The link with GIS needs an intermediate process
- There is no way to create a "selection set" from a statistical process.
- Stored data cannot be easily browsed
- The bridge to other system (SPSS, Lotus) is not supported for recent versions of these systems.

winR+ implementation
The adaptation of the DOS version of the statistical engine requires a full understanding of the software. This obvious statement implies a tremendous investment in time. Up to now, maintenance has been made on a "bug has been found" basis without the control over the entire process. The implementation of new features such as the access to external data source would be extremely difficult if not an impossible job taking into account the human resources of the CELADE Infopal Area.

Proposal for improving the situation

Language

- The number of commands is reduced to a minimum and (hopefully) the language has now a clear object oriented approach
- New functions are incorporated in the language. Furthermore, the architecture of the implementation allows for an easy extension of the language.
- Variable definitions can be stored in the database dictionary. A definition can be called back by a simple reference, thus making the concept of "Computed Type" part of the database dictionary.
- Selection sets can be defined at run time (computed) or defined through LINK files or previously defined SELECTION files

Output

- Output can be directed immediately to a database file. This avoids the GENERATE process of the DOS version.
- Selection sets can be automatically generated and stored in the user's workspace.

Processing

- Data from an other source may be processed as such in the database. We plan to support xBase and Microsoft ACCESS formats.
- The statistical processor execute an "assembled" program. The main task is executed by the syntaxis/ semantic analysis module that generates the code for a virtual machine. This approach opens the way for remote execution of the process and easy transportability of the system.

Maintenance
This is our main concern. The software is modularized and technically documented for the purpose of maintenance and extension of its functionality.

COMMANDS

1. Overview

In addition to the RUNDEF command that define the run environment and which is unique in a command set, only two commands are necessary to perform winR+ a retrieval process: DEFINE and TABLES. A command may expand to several lines.

RUNDEF specifies the general environment of the statistical process. DEFINE declares non dictionary stored variables referred to in the command set and TABLE declares the required output.

Command summary

RUNDEF <runid1>
RUNTITLE <vardef>
SELECTION <selectionset>
UNIVERSE <boolean_expr>
RUNSELECT <**CASES** | **SAMPLE** > <number> <entid1> [**FROM** <entid2>]
PRINTCODE [< **ALL** | **NONE** >] [**SOURCE**] [**PASS1**] [**PASS2**] [**PARSE**] [**MAP**]
OUTPUT [**ASCII** [**FILENAME** <filename>]] | **PRINTER**
USERID <userid > < password >

DEFINE <varid1>
AS <vardef>
FOR <logical_expr>
TYPE <vartype>
RANGE <range_list>
NOTAPPLICABLE <range>
MISSINGVALUE <range>
VARLABEL <string>
VALUELABELS <valuelabel_list>
[**LIKE** <varid2>] (ignored in this version)

TABLE <tableid>
AS <tabletype>
OF <varid_list1> [**BY** <varid_list2> [**BY** <varid_list3> [**BY** <varid_list4>]]]
TITLE <string>
COMPUTEOPTIONS <computeoption_list>
OUTPUTOPTIONS <outputoption_list>
OUTPUTFORMAT <outputoption_list>
PRESENTATIONOPTIONS <presentationoption_list>
AREABREAK <entid_list>

General syntaxis rules

1. In the syntaxis description, a list of tokens is indicated by the token trailed by "list". For example, <varid_list> means a list of <varid>. A list must contain at least one element. Elements are separated by commas. An element must be enclosed in parenthesis whenever itself is a list (see RECODE syntax).

2. No abbreviation is allowed to keywords.

3. Any variable referred to in a clause such as FOR or AS must be a variable previously defined, either in the database dictionary or in a previous DEFINE command of the command set.

4. TABLE replaces AVERAGE, FREQUENCIES, CROSSTABS, TABLES and WRITE of the previous syntax.

The command description includes the following basic tokens:

<entid>	entity identifier of the database entity.
<varid>	<entid>.<varidentifier> Please note the dot (".") that indicates the entity membership.
<string>	String constant or string expression.
<stringconstants>	String constants must be encoded in between quotes: "Text string" or 'Text string'
<arithmetic_expr>	Usual arithmetic expression. Please, not that reference to a variable must be done according to the new <varid> definition. However, arithmetic expression may include functions (see Appendix II for supported functions).
<varidentifier>	string of 8 characters maximum, the first one being alphabetical.

2. The RUNDEF command

Syntax

RUNDEF <runid1>
TITLE <vardef>
SELECTION <selectionset>
UNIVERSE <boolean_expr>
RUNSELECT <CASES | SAMPLE > <number> <entid1> **[FROM** <entid2> **]**
PRINTCODE [< ALL | NONE >] [SOURCE] [PASS1] [PASS2] [PARSE] [MAP]
OUTPUT [ASCII [FILENAME <filename> **]] | PRINTER**
USERID <userid > < password >

Clauses

RUNTITLE <vardef> Character string. Run Title heading for every output page.

SELECTION <areaselection> | **SELFILE** <filename> | **LINKFILE** <filename> | **ALL**

<areaselection> area selection as defined in the winR+ workspace
<filename> DOS valid file name

The qualifier **SELFILE** refers to a traditional REDATAM+ selection file, **LINKFILE** refers to a link file as defined by the winR+GIS link tool and **ALL** refers to the entire database.

UNIVERSE <boolean_expr> Overall selection of data. Evaluated at reading time.

RUNSELECT <CASES | SAMPLE > <number> <entid1> **[FROM** <entid2> **]**

PRINTCODE [<ALL|NONE >][SOURCE][SOURCE][PASS1][PASS2][PARSE][MAP]

Program output options

< ALL | NONE >	Either one of these 2. Any subsequent parameter overrides this clause.
SOURCE	Original command set
TABDEF	Table expansion
PASS1	First pass assembly code
PASS2	Second pass assembly code
PARSE	Parsed command before generating the first pass code*
MAP	Map of constants and variables referred to in the command set

* These options will be deactivated in the distributed version

OUTPUT [ASCII [FILENAME <filename> **]] | PRINTER** Output medium

ASCII	Output in DOS ascii . Minimum formatting takes place
PRINTER	Sends output directly to the printer
FILENAME	Sends output to the file <filename> in ASCII format

USERID <userid > < password > Used for restricted databases.

Parameters are provided by the database owner or database manager.

3. The DEFINE command

Syntax

DEFINE <varid1>
AS <vardef>
FOR <logical_expr>
TYPE <vartype>
RANGE <range_list>
NOTAPPLICABLE <range>
MISSINGVALUE <range>
LABEL <string>
VALUELABELS <valuelabel_list>
DEFOPTIONS [SAVE] [OVERIDE]
[LIKE <varid2> **]**

Clauses

<varid1> Must be a new variable definition

AS <vardef>

This clause defines how the variable is evaluated.

<vardef> **[**<arithm_expr> **|**
QUANTIFY <entid> **|**
<recode_expr> **QUANTIFY** <entid> **|**
SUM <varid> **|**
EXTERNAL <definition> **]**

<recode_expr> **RECODE [**<varid> **|** (<arithmetic_expr) **]** <recodeitem_list>

Please note that an arithmetic expression in a RECODE expression **MUST BE** enclosed in parenthesis.

<recodeitem> **[(** <range_list> **=** <integer> **) | ELSE** <integer>**]**

<range> Single or couples of integer, string constants or real constants.
Examples:

1 - 3
1..3
1 TO 3
LOWEST - 3
4 TO HIGHEST
4..HIGHEST

TO is equivalent to - and can be read ' and all values between '
LOWEST maximum value
HIGHEST minimum value

The symbol " **-** " has been kept for upward compatibility reason. It should be replaced by the mathematical symbol " **..** " which generally used to identify an enumeration.

QUANTIFY <entid> similar to the **QUANTIFY** command of the previous syntax.

SUM <varid> calculates the sum a variable of a lower entity.

EXTERNAL [OVERRIDE] < DBF|MDB|ASCII|TEXT > <external specifications>

The EXTERNAL qualifier associates a variable from an external source of the winR+ database. The OVERRIDE clause informs the system that the variable is being re-defined. The external source specifications depends on the type of the external source:

DBF source files

DBF <xbasefile> **[INDEX <<** indexfilename> | **CREATE >]**
KEYFIELD <keyname> **[UPDATE]**

The variable name as specified in the DEFINE must correspond to the field in the dbf file.

If no INDEX is specified, the file is assumed to be "compact" i.e. that the record sequence in the file corresponds to the sequence of the **selection set**. If an INDEX file name is specified, it must related to the KEY field. The qualifier CREATES provides a mean to create or re-creates the index at run time.

The KEYFIELD corresponds to the entity code to which belongs the variable.

The UPDATE keyword indicates that the file may be updated during the statistical process.

Note: Running the statistical process with only DEFINE commands on external source is a mean to create new variables.

MDB source files

MDB <databasename> <tablename> >
[INDEX < < indexname> | **CREATE >]**
KEYFIELD <keyname> **[UPDATE]**

The variable name as specified in the DEFINE must correspond to the field in the MDB table.

If no INDEX is specified, the table is assumed to be "compact" i.e. that the record sequence in the table corresponds to the sequence of the **selection set**. If an INDEX file name is specified, it must related to the specified KEY field. The qualifier CREATES provides a mean to create or re-creates the index at run time.

The KEYFIELD corresponds to the entity code to which belongs the variable.

The UPDATE keyword indicates that the file may be updated during the statistical process.

Note: Running the statistical process with only DEFINE commands on external source is a mean to create new variables.

ASCII source file

ASCII <filename> **[DELIMITED WITH < BLANKS** | <delimiter>]
[SEQUENCE <sequencenumber>]

TEXT source file

TEXT <filename> [DELIMITED WITH < BLANKS | <delimiter>]

Text files are a special type of ASCII files that includes a header specifying the file structure.

GIS source file

GIS <filename>

GIS files are a special type of ASCII files that includes a header.

Fixed format source file

FIXED <filename> <position> <size> KEY <keyposition> <keysize>

FOR < boolean _expr>

Restricts the computation. Value is calculated only when the expression true. Otherwise the variable is assigned the NOTAPPLICABLE value.

A boolean expression is always assigned to a variable of type boolean. The type boolean is identical to a type integer for which the value -1 is considered equal to TRUE and any other value is considered FALSE. This is Visual Basic convention. A variable declared as boolean is thus treated as an integer variable except that it appears to have only two values: TRUE and FALSE. The user should be aware of this fact **ONLY** when a FOR clause is evaluated through an arithmetic expression which is (in my opinion) a suicidal practice.

For example, in the expression X = (24 = 3 * 8) , X is assigned the value TRUE (or -1)

TYPE <vartype> This clause defines the type of the variable

<vartype> **[INTEGER | REAL | BOOLEAN | STRING]**

RANGE <range_list>

This clause defines the values the variable can take. From this clause, the minimum and maximum value are deduced. Although optional, this clause should be used in every definition, specially when the variable is to be included in a TABLE definition since if the minimum and maximum value are known before process begins the processing time can be drastically reduced.

NOTAPPLICABLE <range>

This clause define not applicable value of the variable. Please note, that the concept of not applicable value is a range and not a single value as it was before in the previous REDATAM+ syntax.

MISSINGVALUE <range>

This clause define missing value of the variable. Please note, that the concept missing is a range and not a single value as it was before in the previous REDATAM+ syntax.

NOT APPLICABLE, MISSING VALUE are processed as the first available out of range value. For instance, if a variable is defined between o and 9, the missing value will be assigned 10 and not applicable value will be assigned 11. The user should not bother with these internal assignment, since these two special categories will never be shown with their numerical values. Please, note that an out of range value in the stored data is assumed to be not applicable. This facility is supported for historical reasons. Actually an out of range value is a constraint violation of the database integrity. These concepts emphasize the importance for the user to have an absolute control over the database content and consequently, over the importance of the RANGE definitions. A special TABLE command has been created (see CODEBOOK keyword in TABLE/AS clause) for the user to check the integrity of the database.

VARLABEL <stringconstant> The variable label

VALUELABELS <valuelabel_list>

A <valuelabel> associates a variable value with a label string. This clause is not mandatory. Value labels are used by default as captions in TABLE outputs. Please note that several value may be associated with the same value label. Example:

(1 "Male") (2 "Female")
(0-15 "Minors") (16 to HIGHEST "Adults")

Please note:

1. list does not contain any ",".
2. Ranges may be assigned a value label

[LIKE <varid2>**]** (ignored in this version)

This clause is an inheritance clause. All attributes of <varid2> are assigned to the currently defined variable. These attributes include the type, label, ranges and value label. The inherited attributes may be overridden by an other clause such as LABEL. <variid2> must be a variable previously defined, either in the database dictionary or in a DEFINE command.

<varid2> is of type <varid>

DEFOPTIONS [SAVE] [OVERRIDE]

SAVE Save the definition in the database directory

OVERIDE Overrides a variable definition of the dictionary. This option applies only for computed variables

Notes:

As suggested, we are studying the possibility to include the valuelabel definition within the syntax of a RECODE statement. The proposal would look like:

RECODE oldvar (1-3,5 = 1 'POOR') (4=2 'MIDDLE CLASS') (6-8 = 3 'RICH')

Examples

The following examples of DEFINE commands assumes the existence in the database dictionary of the entities "ed", "hhold" and "person". The variables "sex", "age", "activity" and "relat" are members of the entity "person" and the variables "water", "elect" and "rooms" are members of the entity "hhold". Indents have no semantic signification and should be used to clarify the presentation.

(Note: the continuation sign ';' does not seem to be necessary. However, in the first stage it bloody simplifies the programming. I shall what I can do....)

1. Defining the sex of household head:

```
DEFINE hhold.sexhh AS person.sex FOR person.relat = 1
        LIKE person.sex RANGE (0-3) VALUELABEL 0 "Missing Value"
```

2. Defining dependents by age groups:

```
DEFINE person.activeagegroup
        AS RECODE person.age (0-14=1) (15-65=2) ELSE 3
        TYPE INTEGER RANGE (1-3) LABEL "Age Group"
        VALUELABELS   1 "Dependent children"
                      2 "Active adults"
                      3 "Dependent adults"
```

Note: The TYPE clause would not be required if we assume the RECODE returns an integer. But what about recoding to a decimal variable.

3. Defining household by presence of male children:

```
DEFINE hhold.malechildren
        AS QUANTIFY person ;
        FOR person.age <= 15 AND person.sex = 1
        TYPE INTEGER
DEFINE hhold.malechildrenpresence
        AS RECODE person.malechildren ( 0 = FALSE ) ELSE TRUE
        TYPE BOOLEAN
        VALUELABEL FALSE "No male children"  TRUE "Male children present"
```

4. Computing an index of poverty at ed level

```
DEFINE hhold.waterindex AS RECODE hhold.water (1-2=1) ELSE 0
DEFINE hhold.powerindex AS RECODE hhold.elect (1-2,4=1) ELSE 0
DEFINE hhold.overcrowd
        AS RECODE ( COUNT(person) / hhold.rooms) (0-2.3=1) ELSE 0
DEFINE hhold.active AS QUANTIFY person
        FOR person.activity IN (1-2,7,9)                    'New logical operator
DEFINE hhold.depratio
        AS RECODE (hhold.active / COUNT(person)) (0.5 TO HIGHEST=1) ELSE 0
DEFINE hhold.index
        AS hhold.overcrowd + hhold.waterindex + hhold.power + hhold.depratio
DEFINE ed.index AS SUM (hhold.index)/COUNT(hhold)
DEFINE ed.indexclass AS RECODE ed.index  (0-1=1) ELSE 2
```

Note: The function COUNT returns the total number a instances of the specified entities belonging to the current variable entity. The QUANTIFY command does the counting as the process goes on. The returned value by a COUNT is the same as the QUANTIFY if no FOR clause modifies the counting.

5. Average age of person in a entity

DEFINE ed.agem AS SUM(person.age)/COUNT(person)

Implementation notes

- Whenever defined through the winR+ dictionary management, the variable definition may be saved permanently in the dictionary (note: a **SAVE** clause in the command set could also be made available) Whenever referred to later a variable definition will automatically be loaded into the command set as a DEFINE statement. prior to its first reference. The user should always be aware that the order of appearance of the DEFINE command is important since any DEFINE command cannot refer to any undefined variable.

- Definition of actual variables (by opposition to Computed variables), i.e. variables whose values have been actually stored on disk should not be changed or if so, extreme care should be taken doing so (note: an **OVERIDE** clause could be implemented to avoid errors)

4. The TABLE command

Syntax

TABLE <tableid>
AS <tabletype>
OF <tablegroup>
TITLE <string>
COMPUTEOPTIONS <computeoption_list>
OUTPUTOPTIONS <outputoption_list>
OUTPUTFORMAT <outputoption_list>
PRESENTATIONOPTIONS <presentationoption_list>
WEIGHT <arith_expr>

Clauses

<tableid> <string>.

Table group identifier. Actually a TABLE command may represent a group of table depending on the OF clause that may imply several tables. In this case each table is identified as an element of an array of tables, the first being tableid(1), the second tableid(2) etc... The identification is made from left to right.

For example, in the command

TABLE mytable AS CROSSTABS OF v1, v2 BY v3, v4

mytable(1) corresponds to v1 BY v3
mytable(2) corresponds to v1 BY v4
mytable(3) corresponds to v2 BY v3
mytable(4) corresponds to v2 BY v4

Note: A list of variables may be enclosed in parenthesis in order to improve clarity:

TABLE mytable AS CROSSTABS OF (v1, v2) BY (v3, v4)

AS **[FREQUENCY | AVERAGE | CROSSTABS | DISTRIBUTION | AREALIST]**

OF <tablegroup>

A table group syntax depends on the type of table required (see AS clause). The following tables describes each case:

AS clause	OF clause syntax
FREQUENCY	<varid_list> **[AREABREAK** <entid_list> **]**
AVERAGE	<varid_list1>**[BY** <varid_list2> **[BY** ... **[BY** ... **]]] [AREABREAK** <entid_list> **]**
CROSSTABS	<varid_list1> **BY** <varid_list2> **[BY** ...**[BY** ... **]] [AREABREAK** <entid_list> **]**
DISTRIBUTION	<varid_list> (*)
AREALIST	<entid> **[,** <varid_list> **]** (*)
CODEBOOK	<varid_list> **[UPDATE]**

The DISTRIBUTION clause list by area the values of a variable. This clause replaces the CROSSTAB of the previous syntax that involved an entity. Only variables of TYPE integer may be declared.

The AREALIST clause produces a list by entity of a set of variables members of the entity. All variables must be members of the same entity. This command is used to extract data from the database and move it to a foreign system.

The CODEBOOK clause is used mainly to get a variable description, including the range of the variable. The UPDATE qualifier corrects eventual consistencies related to the variable.

WEIGHT <arith_expr>

Weight of every count in the tabulation process.

THE STATISTICAL PROCESSOR

1. Overview

A statistical process is dynamically similar for every case of information retrieval. It is mainly a loop (database retrieval main loop) through all the selected database areas (not necessarily geographic) and as this recurrent process goes along, information is picked up from permanent storage, new data is computed and assembled in output items. At the end of this loop, the output formatting takes place. The new statistical process is assembled as a program with specific instructions run by an executor that acts as a virtual computer processor.

The main stages of the statistical process are:

- the syntaxis and semantic analysis of the user's code
- the assembly of the environment and the program code
- the execution of the database main loop
- the output formatting

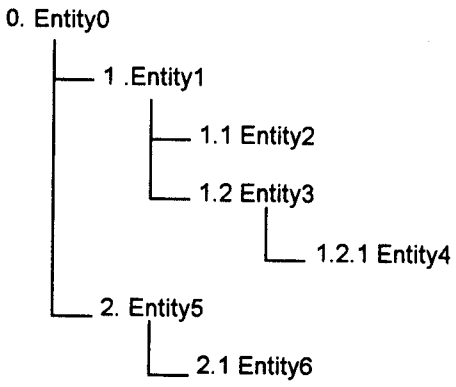
The function of **syntaxis/semantic analysis** of the DEFINE and TABLE commands is to provide the statistical process with all the data elements that have to be taken care of in the process:

- constants
- entities
- variables
- output items

When no error has been encountered, the process provides all the assembly of the "first pass" code for computing the variables. This is mainly the code to be grafted on the database retrieval main loop.

The **assembly of environment and the program code** prepare the program assembly code and the memory mapping of data items to be processed. It performs its task in two passes. The first pass generates the first version of the program at the level of macro instructions. The second pass generate the actual "executable" code. The first pass code contains macro-instructions of the memory mapping and the program itself. The program is built from the database retrieval main loop describe below.

The **database retrieval main loop** is the skeleton onto which the user's specific requirements such as variable calculations and tabulations are grafted. This skeleton reflects the database structure. For example a database with the following structure:



would correspond to the following retrieval main loop:

```
Start Loop Entity0
        Start Loop Entity1
                Start Loop Entity2

                End Loop Entity2
                Start Loop Entity3
                        Start Loop Entity4

                        End Loop Entity4
                End Loop Entity3
        End Loop Entity1
        Start Loop Entity5
                Start Loop Entity6

                End Loop Entity6
        End Loop Entity5
End Loop Entity0
```

The pseudo-code Start Loop and End Loop represents markers of a loop processing. Each loop should be replaced by the following pseudo-code:

Start Entity Loop:

- Initialize entity related temporary variables (if any)
- Initialize pointers of variables to be read (if any)
- **Do**

  Read set of entity variables (if any)

  - Performs computations related variables of entity (pre process if any)

  ```
  Select Case CurrentSelectedEntity
          Case Ent1
                  Start inner loop1 of entity Ent1
                  .............
                  End inner loop 1 of entity Ent1
          Case Ent2
                  Start inner loop 2
                  .........
                  End inner loop 2
  End Select
  ```

  - Performs computations related variables of entity (post process if any)
  - Performs tabulations (if any)
  - Performs intermediates output such as AREABREAK (if any)

  Until end of area

End Entity Loop

- The Loop control is driven by the start and end of the current processed area. The loop is built on a DO/UNTIL scheme. Is assumed that at least one area exists.
- The code generated includes only the referenced entities, i.e. entities that are explicitly referred to such as in TABLE/AREABREAK or TABLE/AREALIST or through the reference of any variable appearing in a DEFINE or TABLE command.
- The Select Case structure selects the lower entity of the current selected sub area

The database main loop is executed for each area selected by the user in the **SELECTION** clause.

```
Initialize selected area List
Do
        Read next selected area
        Process main loop
Until end of selected area list
```

2. Syntaxis/semantic analysis

This module major function is to check the syntaxis and semantic of the user's code, provide error eventually and generate the first pass assembly code.

The first step is taken care of by the module **wrpDEFTABCompile**. This module isolates commands and clauses and update the VarDefs() and TabDefs() arrays as well as the list of variables and constants involved in the process (see Global String Array Storage()) and the necessary first pass code to compute all expressions involved in the commands as well as the database main loop (see Global String Array GenProg()).

Program is assembled in first pass language assembly (see Appendix 5 and Appendix 7).

Call

Flag = **wrpDEFTABCompile** (DefTab$, RunDef As WRPRUN, VarDefs as WRPDEFINE,
TabDefs as WRPTABLE, ErrorMess$)

Parameters

Flag is True when no error has been encountered, False when an error has been encountered.
DefTab$ user's command set.
RunDef definition parameters
VarDefs() contains the parsed DEFINE commands
TabDefs() contains the parsed TABLE commands
ErrorMess contains an error message or blank if no error

Syntax error report may be printed. Any error in a clause will be mentioned with the clause replaced with %nnn (error number) followed by the error message related to the clause. Whenever an unrecognized command or clause is encountered, scanning stops and the ErrMess$ contains the error message.

Programming notes

First, this routine isolates the commands, then calls three main modules for each case: **wrpDefTabRUNDEF**, **wrpDefTabDEFINE**, **wrpDefTabTABLE**. If no error has interrupted the process, the main loop is assembled by **wrpASSEMBLELoop**. The last step is executed by **wrpGENERATECode**. This routine assemble the "executable" version of the program. It also checks the execution parameters such as the availability of database files.

Global Arrays

Storage()

GenProg()

VarDefs() type WRPDEFINE (see below)

TabDefs() type WRPTABLE (see below)

Structures

```
Type WRPRUN
        CommandSet As String
        Exist As Integer
        Command As String
        Error as Integer
        LineNumber As Integer
        NLines As Integer
        JobId As String
        TITLE As String
        SELECTIONClause As String
        UNIVERSEClause As String
        RUNSELECTClause As String
        PRINTCODECLause As String
        OUTPUTClause As String
        USERIDClause As String
        PRINTCODEANY As Integer
        PRINTCODESOURCE As Integer
        PRINTCODEMAP As Integer
        PRINTCODEPARSE As Integer
        PRINTCODEPASS1 As Integer
        PRINTCODEPASS2 As Integer
        OUTPUTTYPE As String
        OUTPUTFILENAME As String
End Type

Type WRPDEFINE
        Command As String
        LineNumber As Integer
        NLines As Integer
        Error As Integer
        VarId As String
        VarName As String
        EntName As String
        ASClause As String
        FORClause As String
        TYPEClause As String
        RANGEClause As String
        LabelClause As String
        VALUELABELSClause As String
        LIKEClause As String
        NOTAPPLICABLEClause As String
        MISSINGVALUEClause As String
        ASTYPE As String
        ASEXPR As String
        ASEXPRCODE As Integer
        ASPARAM As String
        ASPARAMCODE As Integer
        PREPROCESSING As Integer
        Dependency As String
        COMPUTATIONLEVEL As String
End Type
```

```
Type WRPTABLE
        Command As String
        Error As Integer
        LineNumber As Integer
        NLines As Integer
        TableId As String
        ASCLause As String
        OFClause As String
        LABELClause As String
        COMOPTClause As String
        OUTOPTClause As String
        PRESOPTClause As String
        WEIGHTClause As String
        TABLETYPE As String
        AREABREAKClause As String
        DIM1 As String
        DIM2 As String
        DIM3 As String
        DIM4 As String
        TOTDIM As Integer
        COMPUTELEVEL As String
        ALLGROUPS As String
        OUTPUTLEVEL As String
End Type
```

Error Messages

Messages returned in ErrorMess$

000 Empty command set
001 Unexpected ';'
002 Unrecognized command name
003 Unexpected continuation character

005 Illegal variable identifier"
006 No variable definition"
007 Missing AS Clause"
008 Cannot re-define a dictionary variable"
009 Variable already defined"
010 Reference to an unknown entity"
011 Reference to an unknown entity in an AS/QUANTIFY clause"
012 Syntax Error in an AS/RECODE clause"
013 Expected variable identifier in an AS/RECODE clause"
014 Expected variable identifier in an AS/RECODE clause"
016 Illegal variable name found in an AS/RECODE clause"
017 Illegal type declared in a TYPE clause"
018 Illegal string declaration in a LABEL clause"
019 Illegal variable name found in an LIKE clause"
020 Error in an AS/RECODE list"
021 Error in an AS/RECODE expression"
022 Error in an AS/RECODE range affectation list"
023 Error in an AS expression"
024 Error in a FOR clause"
025 Reference is made to an entity of a different branch
026 Reference is made to an illegal entity in a QUANTIFY clause
027 Reference is made to lower entity in computation

201 Unbalanced parenthesis in expression"
202 Left side of a 'TO' key word is missing"
203 Right side of a 'TO' key word is missing"
204 Right term of '+' is missing"
205 Right term of '-' is missing"
206 Missing term in a range affectation"
207 Multiple range affectation operator"
208 Missing element in a list"
209 Missing left term of " & OpCode
210 Missing right term of " & OpCode
211 Missing right term of " & OpCode
212 Unexpected left term of an ELSE keyword"
213 Missing right term of an ELSE keyword"

3. Execution loop

Call

Function **wrpExecute** (Main_Entry as Integer, ErrorMess as string)

Parameters

[To be included in a later version of this working document].

Variables

[To be included in a later version of this working document].

Structures

```
Type WRPVARIABLE
   VarName As String
   EntName As String
   Type As String            'INTEGER | STRING | BOOLEAN
   Location As String         'DISK | COMPUTED
   Label As String
   Documentation As String
   Branch As Integer
   Position As Integer       'Field Position for ASCII files
   AsciiSize As Integer      'Size in ascii representation
   Size As Integer           'Depends on the type
   Decimals As Integer
   Level As Integer
   NotApplicableValue As String
   MissingValue As String
   VarNumber As Integer
   Database As String        'database name or directory
   DatabaseType As String     'ASCII DBF MDB RPLUS
   StreamName As String       'table name or file name
   Categories As Integer
   Security As String        'level of security
   Definition As String      'Dynamic definition of variable
   Dependency As String
   CurrentSlot As Long
   LValue As Long
   DValue As Double
   SValue As String
   WindowSize As Integer
   WindowsInBuffer As Integer
   LeftWindow As Long
   Buffer As String
End Type
Dim VarArray() As WRPVARIABLE
```

```
Type WRPENTITY
   EntName As String
   Label As Integer
   EntCodeProfile As String
   EntNumber As Integer
   EntPos As Integer
   EntSize As Integer
   Branch As Integer
   Level As Integer
   Preorder As Integer
   Parent As Integer
   Child As Integer
   Brother As Integer
   Database As String          'database name or directory
   DatabaseType As String      'ASCII DBF MDB RPLUS
   StreamName As String        'table name or file name
   RefCode As String
   RefLabel As String
   Security As String
   CompoundedCode As String
   StartSlot As Long
   NumberOfSlot As Long
   CurrentSlot As Long
End Type
Dim EntArray()  As WRPENTITY
```

Definition of a frequency table

```
Type WRPTABLEFREQ
        TABLEID As String
        TABLETYPE As String
        TITLE As String
        OUTOPTClause As String
        PRESOPTClause As String
        IFWEIGHT as Integer          'TRUE if variable has a weight
        WEIGHT as Integer            'Variable index in VarArray (refers to the variable weight)
        VARINDEX as integer          'Variable index
        MEMOBANK as integer          'One of the 4 arrays that contains data
        OFFSET as Long               'Offset of the table in MEMORYBANK
End Type
```

Expansion of instruction TAB_FREQ

```
****** Cell index computation **********************************
TbIndex = Prog(PCounter+1)
VarIndex = FreqArray(TbIndex).VARINDEX
CellIndex = VarArray(VarIndex).LValue + FreqArray(TbIndex).OFFSET

****** Bank n Affectation **********************************
MEMOBANK1(CellIndex) = MEMOBANK1(CellIndex) + 1
```

Note:

In this example the tabulation array had been set to MEMORYBANK1
CellIndex is a register variable of the execution engine

```
****** Virtual memory affectation **********************************
VIRTUALMEMO
```

```
Type WRPTABLEFREQ
        TableId As String
        TABLETYPE As String
        TITLE As String
        OUTOPTClause As String
        PRESOPTClause As String
        WEIGHT as Integer            'Variable index in VarArray (refers to the variable weight)
        DIM1 As Integer              'Table dimensions DIM1 to DIM4
        DIM2 As Integer              ' According to the total number of dimension
        DIM3 As Integer              ' index is computed
        DIM4 As Integer
        VARDIM0 as integer                   'Variable index
        VARDIM1 as integer
        VARDIM2 as integer
        VARDIM3 as integer
        VARDIM4 as integer
        MEMOBANK as integer          'One of the 4 arrays that contains data
        OFFSET as Long               'Offset of the table in MEMORYBANK
End Type
```

TAB_X1

```
TbIndex = Prog(PCounter+1)
VarIndex1 = TabArray(TbIndex).VARDIM1
OffSetVar = TabArray(TbIndex).DIM1 * (VarArray(VarIndex1).LValue-1)
VarIndex2 = TabArray(TbIndex).VARDIM2
CellIndex = VarArray(VarIndex2).LValue + OffSetVar + TabArray(TbIndex).OFFSET
MEMOBANK1(CellIndex) = MEMOBANK1(CellIndex) + 1
```

TAB_X2, TAB_X3, TAB_X4

AVG_X0
AVG_X1
AVG_X2
AVG_X3
AVG_X4
DISTRIB

Loop Structure

```
Pcounter = Main_Entry
Do While Pcounter > 0
        OpCode = Prog(PCounter)
        Select Case OpCode
                Case ...

                Case

                Case STOPRUN
                        Pcounter = 0
        Ens Select
Loop
If PCounter > 0 Then
        wrpExecute = True
Else
        wrpExecute = False
Endif
Exit Function
```

4. Output processing

[To be included in a later version of this working document].

THE IMPLEMENTATION SCHEME

The new language will be implemented in two ways

- Commands will be entered as a text file.
- Commands will be form driven.

In both case, drag and drop facilities will be provided to facilitate the entry.

APPENDIX 1 - Old vs New syntax

Example 1

R+ syntax

```
SELECTION filename
RECODE AGE TO AGEGROUP (0-14=1)(15-65=2) ELSE 3
FREQUENCY AGEGROUP
IF AGEGROUP = 1 THEN CROSSTABS  AGEGROUP BY SEX
```

winR+ syntax

```
RUNDEF Example SELECTION SELFILE filename
DEFINE person.agegroup AS RECODE person.age (0-14=1)(15-65=2) ELSE 3
TABLE FREQ1 AS FREQUENCY OF person.sex
TABLE CROSS1 AS CROSSTABS OF person.agegroup BY person.sex ;
        FOR person.agegroup = 1
```

Example 2

R+ syntax

```
SELECTION filename
DEFINE HHOLD TOTPERS
FOREACH HHOLD
        QUANTIFY PERSON TO TOTPERS
        IF AND TOTPERS > 4 THEN RELATION = 1 THEN COMPUTE
                        SEXHEADOFHOUSEHOLD = SEX
END
FREQUENCY SEXHEADOFHOUSEHOLD
```

winR+ syntax

```
RUNDEF Example SELECTION SELFILE filename
DEFINE hhold.totperson AS QUANTIFY person
DEFINE hhold.sexofhhhead AS person.sex FOR person.relat = 1
TABLE freq1 AS FREQUENCY OF hhold.sexofhhhead  FOR hhold.totperson > 4 ;
        LABEL "Sex of household head with more than four persons
```

Example 3

R+ syntax:

```
DEFINE REAL HHOLD MINORRATIO
DEFINE HHOLD TOTPERSON, TOTMINOR
FOREACH HHOLD
        QUANTIFY PERSON TO TOTPERSON
        COMPUTE SEXHEADOFHOUSEHOLD = 0
        IF SEX = 1 THEN COMPUTE SEXHEADOFHOUSEHOLD = SEX
        IF AGE < 15 THEN QUANTIFY PERSON TO TOTMINORS
        COMPUTE MINORRATIO = TOTMINORS/TOTPERSON
END
AVERAGE MINORRATIO BY SEXHH
```

winR+ syntax

```
DEFINE ed.minorratio AS RATIO( person) FOR person.age < 15
DEFINE hhold.sexheadofhousehold AS person.sex FOR person.relat = 1
TABLE indic1 AS AVERAGE OF segmento.minorratio BY hhold.sexheadofhousehold
```

APPENDIX 2 - List of language constants, operators & available functions

Constants

PI	3.14159265358979323846
e	2.71828182845904523536
TRUE	-1
FALSE	0

Artithmetic operators

+ - * / \ ^ MOD &

Relational operators

>= <= = > <

Logical Operators

NOT OR AND

Set operator

IN <litem_list>

Mathematical Functions

ARCSINH (<arithmetic_ expr >)
ARCTANH (<arithmetic_ expr >)
ARCCSCH (<arithmetic_ expr >)
ARCSIN (<arithmetic_ expr >)
ARCSEC (<arithmetic_ expr >)
ARCCOT (<arithmetic_ expr >)
TANH (<arithmetic_ expr >)
CSCH (<arithmetic_ expr >)
SEC (<arithmetic_ expr >)
COT (<arithmetic_ expr >)
COS (<arithmetic_ expr >)
ATN (<arithmetic_ expr >)
EXP (<arithmetic_ expr >)
CLG (<arithmetic_ expr >)
FACT (<arithmetic_ expr >)
TODAY%

ARCCOSH (<arithmetic_ expr >)
ARCSECH (<arithmetic_ expr >)
ARCCOTH (<arithmetic_ expr >)
ARCCOS (<arithmetic_ expr >)
ARCCSC (<arithmetic_ expr >)
SINH (<arithmetic_ expr >)
SECH (<arithmetic_ expr >)
COTH (<arithmetic_ expr >)
CSC (<arithmetic_ expr >)
SIN (<arithmetic_ expr >)
TAN (<arithmetic_ expr >)
LOG (<arithmetic_ expr >)
SQR (<arithmetic_ expr >)
ABS (<arithmetic_ expr >)
VAL (<string_expr>)

String Function

STR (<arithmetic_ expr >)
TODAY$

REDATAM specific functions

RECODE (<arithmetic_expr>) <recode_list>
COUNT (<entid>)

(Author's Note: I don't know who (the hell) is going to use ARCSINH in a REDATAM command set, but I have put it here for the sake of the theoretical exercise)

APPENDIX 3 - Error Messages Summary

[To be included in a later version of this working document].

APPENDIX 4 - Internal structures

Structures

Type **VARIABLEDEF**

VarName as string	Variable identifier
Entity as string	Entity to which the variable belong to
Type as integer	Type of variable
ValueS as string	
ValueD as double	
ValueI as Long	
RangeMin as single	
RangeMax as single	
LineNumber as Integer	Line number in the user's command set
As as string	As clause as defined by the user
AsProg as integer	Index of the expression calculation
For as string	For Clause as defined by the user
ForProg as integer	Index of the expression calculation
LabelPointer as integer	Pointer to StringArray()
ValueLabelStart as integer	Pointer to StringArray() of first Valuelabel
ValueLabelLen as integer	Number of Value Labels

End type

Type **ENTITYDEF**

End Type

Type **OUTPUTTABLEDEF**

End Type

Type **SELECTIONAREA**

TotalNumber as integer
Current as integer

End Type

Variables and Constants

Constant

SPC_DEFINE	= 1
SPC_TABLE	= 2

Variables

NSPCLine as Integer	Number of commands
SPCLine() as String	Command text
SPCComType() as Integer	Command type (SPC_DEFINE, SPC_TABLE)
NStringArray as Integer	Length of string array
StringArray() as String	General purpose string array to store string members
NBVarDef as Integer	Number of loaded dictionary variables
BVarDef() as VARIABLEDEF	Array of dictionary variables
NLVarDef as Integer	Number of locally defined variables
LVarDef() as VARIABLEDEF	Array of locally defined variables
SRegister() as Single	Register array of type Single
Dregister() as Double	Register array of type Double
IRegister() as Integer	Register array of type Integer
LRegister() as Long	Register array of type Long
Sregister() as String	Register array of type String
Cregister() as Long	Register array of type counter

APPENDIX 5 - First Pass Assembly Language

The first pass assembly language reflects the main loop process and refers to variables, entities, tableidelement and area selection list. Each line contains the address section, the instruction symbol and the list of parameters for the related instruction. The address is a unique symbol automatically generated whenever there is a need for it, such as in a jump instruction. Examples are included to illustrate the language. The following list describes the parameter list for each instruction.

Area selection instructions

INIT_AREA_LIST		'Initialize reading of area selection
READNEXT_AREA	address	'Read next area, if not at end, jump to address

Entity related instructions

INIT_ENT_LOOP	<entid1>,<entid2>	'Initiate reading of child entity entid2 in entity1 loop
READNEXT_ ENT	<entid>, address	'Skip to next entity, jump to address if OK

Variable related instructions

READINIT_VAR	<varid>	'Load next variable value from disk into memory
READNEXT_VAR	<varid>	'Load next variable value from disk into memory
COMPUTE_FOR_VAR	<varid>	'Evaluate FOR expression for a variable
JMP_ON_FALSE	< varid >, address	'Jump to address if FOR is **FALSE**
COMP_RECEXPR	<varid>	Compute DEFINE/AS clause expression
COMP_REC_RANGE	<varid>	Compute DEFINE/AS clause expression
COMP_QTFY	<varid>	Compute DEFINE/AS clause expression
MOVEVALUE		
COMP_SUM		
COMP_EXPR		

Tabulation related instructions

COMPUTE_FOR_TAB	<tabidelement>	'Evaluate FOR expression for a variable
JMP_ON_FALSE	<tabidelement>, address	'Jump to address if FOR is **FALSE**
TAB_FREQ	<tableidelement>	
TAB_X1 to TAB_X4	<tableidelement>	
AVG_X0 to AVG_X4	<tableidelement>	
DISTRIB	<tableidelement>	
OUTTAB_FREQ	<tableidelement>	'Save table element output
OUTTAB_X2 to OUTTAB_X4	< tabidelement >	
OUTAVG_X0 to OUTAVG_X4	< tabidelement >	
OUTDISTRIB	<tableidelement>	
OUTAREALIST	<tableidelement>	
OUTCODEBOOK	<tableidelement>	

Other instructions

ENTRY		'No operation, identify entry point
NOOP		'No operation, used to specify branch address
STOPRUN		'End of program
RTJ	<define_id>	'Return jump to a procedure
JMP		

This first pass assembly code is the source for generating the second pass assembly code or 'executable' code. Each operation is expanded in a linear way.

APPENDIX 6 - Second Pass Assembly Language

Operation code expansion

First pass code	Expansion
INIT_AREA_LIST	INIT_AREA_LIST
READNEXT_AREA	READNEXT_AREA
INIT_ENT_LOOP	INIT_ENT_LOOP
READNEXT_ ENT	READNEXT_ENT
READINIT_VAR	OPENBINFILE
READNEXT_VAR	READBINLONG
	READBINREAL
	READBINSTRING
COMPUTE_FOR_VAR	**********
JMP_ON_FALSE	**********
COMP_RECEXPR	**********
COMP_REC_RANGE	**********
COMP_QTFY	**********
MOVEVALUE	
COMP_SUM	**********
COMP_EXPR	**********
COMPUTE_FOR_TAB	**********
JMP_ON_FALSE	**********
TAB_FREQ	CELLCNT_FREQ
	TABMBx
TAB_X1 to TAB_X4	**********
AVG_X0 to AVG_X4	**********
DISTRIB	**********
OUTTAB_FREQ	**********
OUTTAB_X2 to OUTTAB_X4	**********
OUTAVG_X0 to OUTAVG_X4	**********
OUTDISTRIB	**********
OUTAREALIST	**********
OUTCODEBOOK	**********

Other instructions

ENTRY	ENTRY
NOOP	NOOP
STOPRUN	STOPRUN
RTJ	**********
JMP	JMP

APPENDIX 7 - Operation code Summary

Symbols:

paddress	integer value representing the program pointer in GenProg()
group	operation group 1: first pass assembly, 2: second pass assembly

value	Group	Mnemonic	Syntax	Comments
0	1 2	NOOP		
1	1 2	RET		
2		JMP	paddress	Unconditional jump
3		RTJ	paddress	Return jump
4		READV		
5		LDXL		
6		LDXD		
7		LDXB		
8		LDXS		
9		MOVX		
10		STXL		
11		STXD		
12		STXB		
13		STXS		
14		REC		
30		MULT		
31		DIV		
32		IDIV		
33		MDLO		
34		ADD		
35		SUBS		
36		ADD0		
37		SUB0		
38		GT		
39		LT		
40		GE		
41		LE		
42		EQ		
43		ANDOP		
44		OROP		
45		EXPN		
46		RANGE		
47		RANGEDEF		
48		STOPRUN		

value	Group	Mnemonic	Syntax		Comments
60		INIT_AREA_LIST			
61		READ_NEXT_AREA			
62		INIT_ENT_LOOP			
63		INIT_CHILDENT_LOOP			
64		READ_NEXT_ENT			
65		READ_NEXT_VAR			
66		COMPUTE_FORV			
67		COMPUTE_FORT			
68		COMPUTE_REC_EXP			
69		CON_COMP_REC_RANGE			
70		UNC_COMP_REC_RANGE			
74		TABULATE			
75		OUTABLE			
77		MOVEVALUE			
78		ENTRY			
81		CON_COMP_EXPR	<varid>		C/Compute DEFINE/AS clause expression
82		CON_COMP_REC	<varid>		C/Compute DEFINE/AS clause expression
83		CON_COMP_QTFY	<varid>		C/Compute DEFINE/AS clause expression
84		UNC_COMP_EXPR	<varid>		Compute DEFINE/AS clause expression
85		UNC_COMP_REC	<varid>		Compute DEFINE/AS clause expression
86		UNC_COMP_QTFY	<varid>		Compute DEFINE/AS clause expression
101		ARCSINH			
102		ARCCOSH			
103		ARCTANH			
104		ARCSECH			
105		ARCCSCH			
106		ARCCOTH			
107		ARCSIN			
108		ARCCOS			
109		ARCSEC			
110		ARCCSC			
111		ARCCOT			
112		SINH			
113		TANH			
114		SECH			
115		CSCH			
116		COTH			
117		SEC			
118		CSC			
119		COT			
120		SINF			
121		COSF			
122		TANF			
123		ATNF			
124		LOGF			
125		EXPF			
126		SQRF			
127		CLGF			
128		ABSF			
129		NOTF			
130		FACT			
131		VALI	'conversion string to integer		
132		VALL	conversion string to integer		
133		VALD	'conversion string to integer		

RECODE operation

```
PCounter        RECODE
PCounter+1      <rangedefid >

Type WRPRANGEDEF
        RangeType           Long, String or Double
        RangeDefaultType    1: Keep original Value, 2: Take Value defined
        OffSet              Index in Range array
        Size                Total number of Ranges
        DefaultL
        DefaultD
        DefaultS
End Type
RangeDefArray()

Type WRPRANGE
        RangeOperation      1: Min to Max  2: Min is open  3: Max is open  4:Single value
        LValueMin
        LValueMax
        DvalueMin
        DValueMax
        Label
        LRange
        DRange
        SRange
End Type
RangeArray()
```

APPENDIX 8 - List of Reserved Words

[To be included in a later version of this working document].