A NEW OPEN STRUCTURE AND LANGUAGE FOR THE
\texttt{winR+} STATISTICAL PROCESSOR

(Working Paper)
# TABLE OF CONTENTS

**BACKGROUND (1)**
- Present situation
- A better way of doing things

**COMMANDS (3)**
- Overview
- The RUNDEF command
- The DEFINE command
- The TABLE command

**STATISTICAL PROCESSOR (15)**
- Overview
- Syntax/Semantic analysis
- Execution Loop
- Output processing

**IMPLEMENTATION SCHEME (29)**
- Prototype
- Final version

**APPENDIX 1 - Old vs New syntax, few examples (30)**

**APPENDIX 2 - Constants, Operator & Functions (31)**

**APPENDIX 3 - Error Messages (33)**

**APPENDIX 4 - Internal structures summary (34)**

**APPENDIX 5 - First Pass Assembly Language (36)**

**APPENDIX 6 - Second Pass Assembly Language (38)**

**APPENDIX 7 - Operation code Summary (39)**

**APPENDIX 8 - List of Reserved Words (42)**

This document has been produced by Serge Poulard. The opinions expressed in the document are not necessarily those of the Organization.
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>
  RUNITITLE <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 syntax rules

1. In the syntax 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 <valuellabel_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> | (<arithm_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**

```plaintext
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 CREATE 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**

```plaintext
MDB <databasesname> <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 CREATE 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 0 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. <varid2> 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 OVERRIDE 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 lists 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 syntax 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 syntax/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:

```
0. Entity0
  1. Entity1
    1.1 Entity2
    1.2 Entity3
    1.2.1 Entity4
  2. Entity5
    2.1 Entity6
```
would correspond to the following retrieval main loop:

Start Loop Entity0
   Start Loop Entity1
      Start 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 loop 1 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. It 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's 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 6 and Appendix 7).

Call

\[ \text{Flag} = \text{wrpDEFTABCompile}(\text{DefTab$}, \text{RunDef As WRPRUN}, \text{VarDefs as WRPDEFINE}, \text{TabDefs as WRPTABLE}, \text{ErrorMess$}) \]

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
- Jobld As String
- TITLE As String
- SELECTIONClause As String
- UNIVERSEClaus e 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
- Vari ld As String
- VarName As String
- EntName As String
- ASClause As String
- F RClause As String
- TYPEclause As String
- RANGEclause As String
- LabelClause As String
- VALUELABELClause 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
  PREOPTClause As String
  IFWEIGHT as Integer
  WEIGHT as Integer
  VARINDEX as Integer
  MEMOBANK as integer
  OFFSET as Long
  'TRUE if variable has a weight
  'Variable index in VarArray (refers to the variable weight)
  'Variable index
  'One of the 4 arrays that contains data
  'Offset of the table in MEMORYBANK
End Type

Expansion of instruction TAB_FREQ

****** Cell index computation ********************
TblIndex = Prog(PCounter+1)
VarIndex = FreqArray(TblIndex).VARINDEX
CellIndex = VarArray(VarIndex).LValue + FreqArray(Tblindex).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
  PREOPTClause As String
  WEIGHT as Integer
  DIM1 As Integer
  DIM2 As Integer
  DIM3 As Integer
  DIM4 As Integer
  VARDIM0 as integer
  VARDIM1 as integer
  VARDIM2 as integer
  VARDIM3 as integer
  VARDIM4 as integer
  MEMOBANK as integer
  OFFSET as Long
  'Variable index in VarArray (refers to the variable weight)
  'Table dimensions DIM1 to DIM4
  'According to the total number of dimension
  'index is computed
  'Variable index
  'One of the 4 arrays that contains data
  '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
DISTRIJ
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 hhhold.totperson AS QUANTIFY person
DEFINE hhhold.sexofhhhead AS person.sex FOR person.relat = 1
TABLE freq1 AS FREQUENCY OF hhhold.sexofhhhead FOR hhhold.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 hhhold.sexheadofhousehold AS person.sex FOR person.relat = 1
TABLE indic1 AS AVERAGE OF segmento.minorratio BY hhhold.sexheadofhousehold
APPENDIX 2 - List of language constants, operators & available functions

Constants

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI</td>
<td>3.14159265358979323846</td>
</tr>
<tr>
<td>e</td>
<td>2.71828182845904523536</td>
</tr>
<tr>
<td>TRUE</td>
<td>-1</td>
</tr>
<tr>
<td>FALSE</td>
<td>0</td>
</tr>
</tbody>
</table>

Arithmetic operators

+ - * / \ ^ MOD &

Relational operators

>= <= = > <

Logical Operators

NOT OR AND

Set operator

IN <item_list>

Mathematical Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCSINH</td>
<td>arcsinh</td>
</tr>
<tr>
<td>ARCTANH</td>
<td>arctanh</td>
</tr>
<tr>
<td>ARCCSCH</td>
<td>arccsch</td>
</tr>
<tr>
<td>ARCSIN</td>
<td>arcsin</td>
</tr>
<tr>
<td>ARCSEC</td>
<td>arcsec</td>
</tr>
<tr>
<td>ARCCOT</td>
<td>arccot</td>
</tr>
<tr>
<td>TANH</td>
<td>tanh</td>
</tr>
<tr>
<td>CSCH</td>
<td>csch</td>
</tr>
<tr>
<td>SEC</td>
<td>sec</td>
</tr>
<tr>
<td>COT</td>
<td>cot</td>
</tr>
<tr>
<td>COS</td>
<td>cos</td>
</tr>
<tr>
<td>ATN</td>
<td>atn</td>
</tr>
<tr>
<td>EXP</td>
<td>exp</td>
</tr>
<tr>
<td>CLG</td>
<td>clg</td>
</tr>
<tr>
<td>FACT</td>
<td>fact</td>
</tr>
<tr>
<td>TODAY%</td>
<td>today%</td>
</tr>
<tr>
<td>ARCCOSH</td>
<td>arccosh</td>
</tr>
<tr>
<td>ARCCSECH</td>
<td>arccsech</td>
</tr>
<tr>
<td>ARCCOTH</td>
<td>arccoth</td>
</tr>
<tr>
<td>ARCCOS</td>
<td>arccos</td>
</tr>
<tr>
<td>ARCCSC</td>
<td>arccsc</td>
</tr>
<tr>
<td>SINH</td>
<td>sinh</td>
</tr>
<tr>
<td>SECH</td>
<td>sech</td>
</tr>
<tr>
<td>COTH</td>
<td>coth</td>
</tr>
<tr>
<td>CSC</td>
<td>csc</td>
</tr>
<tr>
<td>SIN</td>
<td>sin</td>
</tr>
<tr>
<td>TAN</td>
<td>tan</td>
</tr>
<tr>
<td>LOG</td>
<td>log</td>
</tr>
<tr>
<td>SQR</td>
<td>sqrt</td>
</tr>
<tr>
<td>ABS</td>
<td>abs</td>
</tr>
<tr>
<td>VAL</td>
<td>val</td>
</tr>
</tbody>
</table>

String Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR</td>
<td>string</td>
</tr>
<tr>
<td>TODAY$</td>
<td>today$</td>
</tr>
</tbody>
</table>

REDATAM specific functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECODE</td>
<td>recode</td>
</tr>
<tr>
<td>COUNT</td>
<td>count</td>
</tr>
</tbody>
</table>

(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
Entity as string
Type as integer
ValueS as string
ValueD as double
ValueL as Long

RangeMin as single
RangeMax as single

LineNumber as Integer
As as string
AsProg as integer
For as string
ForProg as integer

LabelPointer as integer
ValueLabelStart as integer
ValueLabelLen as integer

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 IntegerCommand type (SPC_DEFINE, SPC_TABLE)

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

Entity related instructions

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

Variable related instructions

READINIT_VAR <varid>
READNEXT_VAR <varid>
COMPUTE_FOR_VAR <varid>
JMP_ON_FALSE <varid>, address
'Load next variable value from disk into memory'
'Load next variable value from disk into memory'
'Evaluate FOR expression for a variable'
'Jump to address if FOR is FALSE'

COMPUTE_RECEXPR <varid>
COMPUTE_REC_RANGE <varid>
COMPUTE_QTFY <varid>
MOVEVALUE
COMP_SUM
COMP_EXPR
'Compute DEFINE/AS clause expression'
'Compute DEFINE/AS clause expression'
'Compute DEFINE/AS clause expression'

Tabulation related instructions

COMPUTE_FOR_TAB <tabidelement>
JMP_ON_FALSE <tabidelement>, address
'Evaluate FOR expression for a variable'
'Jump to address if FOR is FALSE'

TAB_FREQ
TAB_X1 to TAB_X4
AVG_X0 to AVG_X4
DISTRIB
OUTTAB_FREQ
OUTTAB_X2 to OUTTAB_X4
OUTAVG_X0 to OUTAVG_X4
OUTDISTRIB
OUTREALIST
OUTCODEBOOK
'Save table element output'

Other instructions

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

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**

<table>
<thead>
<tr>
<th>First pass code</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT_AREA_LIST</td>
<td>INIT_AREA_LIST</td>
</tr>
<tr>
<td>READNEXT_AREA</td>
<td>READNEXT_AREA</td>
</tr>
<tr>
<td>INIT_ENT_LOOP</td>
<td>INIT_ENT_LOOP</td>
</tr>
<tr>
<td>READNEXT_ENT</td>
<td>READNEXT_ENT</td>
</tr>
<tr>
<td>READINIT_VAR</td>
<td>OPENBINFILE</td>
</tr>
<tr>
<td>READNEXT_VAR</td>
<td>READBINLONG</td>
</tr>
<tr>
<td></td>
<td>READBINREAL</td>
</tr>
<tr>
<td></td>
<td>READBINSTRING</td>
</tr>
<tr>
<td>COMPUTE_FOR_VAR</td>
<td>**********</td>
</tr>
<tr>
<td>JMP_ON_FALSE</td>
<td>**********</td>
</tr>
<tr>
<td>COMP_RECEXPR</td>
<td>**********</td>
</tr>
<tr>
<td>COMP_REC_RANGE</td>
<td>**********</td>
</tr>
<tr>
<td>COMP_QTfy</td>
<td>**********</td>
</tr>
<tr>
<td>MOVEVALUE</td>
<td></td>
</tr>
<tr>
<td>COMP_SUM</td>
<td></td>
</tr>
<tr>
<td>COMP_EXPR</td>
<td></td>
</tr>
<tr>
<td>COMPUTE_FOR_TAB</td>
<td>**********</td>
</tr>
<tr>
<td>JMP_ON_FALSE</td>
<td>**********</td>
</tr>
<tr>
<td>TAB_FREQ</td>
<td>CELL_CNT_FREQ</td>
</tr>
<tr>
<td></td>
<td>TABMBX</td>
</tr>
<tr>
<td>TAB_X1 to TAB_X4</td>
<td>**********</td>
</tr>
<tr>
<td>AVG_X0 to AVG_X4</td>
<td></td>
</tr>
<tr>
<td>DISTRIB</td>
<td></td>
</tr>
<tr>
<td>OUTTAB_FREQ</td>
<td></td>
</tr>
<tr>
<td>OUTTAB_X2 to OUTTAB_X4</td>
<td></td>
</tr>
<tr>
<td>OUTAVG_X0 to OUTAVG_X4</td>
<td></td>
</tr>
<tr>
<td>OUTDISTRIB</td>
<td></td>
</tr>
<tr>
<td>OUTAREALIST</td>
<td></td>
</tr>
<tr>
<td>OUTCODEBOOK</td>
<td></td>
</tr>
</tbody>
</table>

**Other instructions**

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>ENTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOOP</td>
<td>NOOP</td>
</tr>
<tr>
<td>STOPRUN</td>
<td>STOPRUN</td>
</tr>
<tr>
<td>RTJ</td>
<td>**********</td>
</tr>
<tr>
<td>JMP</td>
<td>JMP</td>
</tr>
</tbody>
</table>
APPENDIX 7 - Operation code Summary

Symbols:
- \texttt{paddress}: integer value representing the program pointer in \texttt{GenProg()}
- \texttt{group}: operation group 1: first pass assembly, 2: second pass assembly

<table>
<thead>
<tr>
<th>value</th>
<th>Group</th>
<th>Mnemonic</th>
<th>Syntax</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2</td>
<td>NOOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 2</td>
<td>RET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>JMP</td>
<td>paddress</td>
<td>Unconditional jump</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>RTJ</td>
<td>paddress</td>
<td>Return jump</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>READV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>LDXL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>LDXD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>LDXB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>LDXS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>MOVX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>STXL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>STXD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>STXB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>STXS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>REC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>MULT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>DIV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>IDIV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>MDLO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td>ADD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td>SUBS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td>ADD0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
<td>SUB0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
<td>GT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
<td>LT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>GE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
<td>LE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
<td>EQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
<td>ANDOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
<td>OROP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
<td>EXPN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
<td>RANGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
<td>RANGEDEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
<td>STOPRUN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>Group</td>
<td>Mnemonic</td>
<td>Syntax</td>
<td>Comments</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>---------------------</td>
<td>----------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>60</td>
<td></td>
<td>INIT_AREA_LIST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
<td>READ_NEXT_AREA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td></td>
<td>INIT_ENT_LOOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td></td>
<td>INIT_CHILDENT_LOOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td>READ_NEXT_ENT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td></td>
<td>READ_NEXT_VAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td></td>
<td>COMPUTE_FORV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td></td>
<td>COMPUTE_FORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td></td>
<td>COMPUTE_REC_EXP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td></td>
<td>CON_COMP_REC_RANGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td>UNC_COMP_REC_RANGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td></td>
<td>TABULATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td></td>
<td>OUTABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td></td>
<td>MOVEVALUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td></td>
<td>ENTRY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>81</td>
<td></td>
<td>CON_COMP_EXPR</td>
<td>&lt;varid&gt;</td>
<td>C/Compute DEFINE/AS clause expression</td>
</tr>
<tr>
<td>82</td>
<td></td>
<td>CON_COMP_REC</td>
<td>&lt;varid&gt;</td>
<td>C/Compute DEFINE/AS clause expression</td>
</tr>
<tr>
<td>83</td>
<td></td>
<td>CON_COMP_QTFY</td>
<td>&lt;varid&gt;</td>
<td>C/Compute DEFINE/AS clause expression</td>
</tr>
<tr>
<td>84</td>
<td></td>
<td>UNC_COMP_EXPR</td>
<td>&lt;varid&gt;</td>
<td>Compute DEFINE/AS clause expression</td>
</tr>
<tr>
<td>85</td>
<td></td>
<td>UNC_COMP_REC</td>
<td>&lt;varid&gt;</td>
<td>Compute DEFINE/AS clause expression</td>
</tr>
<tr>
<td>86</td>
<td></td>
<td>UNC_COMP_QTFY</td>
<td>&lt;varid&gt;</td>
<td>Compute DEFINE/AS clause expression</td>
</tr>
<tr>
<td>101</td>
<td></td>
<td>ARCSINH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td></td>
<td>ARCCOSH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
<td>ARCTANH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
<td>ARCSECH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
<td>ARCCSCH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td></td>
<td>ARCCOTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>107</td>
<td></td>
<td>ARCSIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td></td>
<td>ARCCOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>109</td>
<td></td>
<td>ARCSEC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td>ARCCSC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td>ARCCOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td></td>
<td>SINH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>113</td>
<td></td>
<td>TANH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>114</td>
<td></td>
<td>SECH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>115</td>
<td></td>
<td>CSCH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>116</td>
<td></td>
<td>COTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>117</td>
<td></td>
<td>SEC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>118</td>
<td></td>
<td>CSC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td></td>
<td>COT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td></td>
<td>SINF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>121</td>
<td></td>
<td>COSF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>122</td>
<td></td>
<td>TANF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>123</td>
<td></td>
<td>ATNF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>124</td>
<td></td>
<td>LOGF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>125</td>
<td></td>
<td>EXPF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>126</td>
<td></td>
<td>SQRF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>127</td>
<td></td>
<td>CLGF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td>ABSF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>129</td>
<td></td>
<td>NOTF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td></td>
<td>FACT</td>
<td></td>
<td>'conversion string to integer</td>
</tr>
<tr>
<td>131</td>
<td></td>
<td>VALI</td>
<td></td>
<td>'conversion string to integer</td>
</tr>
<tr>
<td>132</td>
<td></td>
<td>VALL</td>
<td></td>
<td>'conversion string to integer</td>
</tr>
<tr>
<td>133</td>
<td></td>
<td>VALD</td>
<td></td>
<td>'conversion string to integer</td>
</tr>
</tbody>
</table>
RECODE operation

PCounter  RECODE
PCounter+1 <rangedefid>

Type WRPRANGEDEF
  RangeType
  RangeDefaultType
  OffSet
  Size
  DefaultL
  DefaultD
  DefaultS
End Type
RangeDefArray()

Type WRPRANGE
  RangeOperation
  LValueMin
  LValueMax
  DValueMin
  DValueMax
  Label
  LRange
  DRange
  SRange
End Type
RangeArray()

Long, String or Double
1: Keep original Value, 2: Take Value defined
Index in Range array
Total number of Ranges

1: Min to Max  2: Min is open  3: Max is open  4: Single value
APPENDIX 8 - List of Reserved Words

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