THE GROUPLE TUTORIAL

dav lion

march 1991

This document is intended to explain the grouple doctrine, and to provide code fragments to explain the most commonly used of the bazillion nancy functions available. As the code examples follow our coding standards, it is recommended that you read the code_facism document before continuing this document.

Grouples

Grouples (the word is derived from GROUps of tuPLES, lion '90) are vectors of strings. Individual strings, "words", are delimited by white space, or parentheses. Grouples are delimited by square brackets, literal characters are preceded by '\'.

```
          [ this is a six word grouple]

          [ this (is a three word) grouple]

          [ Mathematica likes (\[lots of \]) brackets]
```

Here is the definition of a grouple, from veos/src/include/grouple.h
Note that grouples know how many words they contain, and that pointer and handle types are defined.

```
          typedef char   *TWord;

          typedef struct {
               int            iWords;
               TWord          *pWordList;
               }       TGrouple,
                     *TPGrouple,
                     **THGrouple;
```

Groupls, being structs, are passed by reference. Thus, primitives accept TPGrouple's. A canonical primitive definition might look like

```
          TVeosErr
          happyPrimitive(pGrouple)
              TPGrouple pGrouple;
          {
             return (VEOS_SUCCESS);

             }/*somePrimName*/
```

A slightly more useful (and less stupidly optimistic) primitive might look like this:

```
          TVeosErr
          countingPrim(pGrouple)
              TPGrouple pGrouple;
          {
            TVeosErr    status = VEOS_FAILURE;

            if (pGrouple) {           /*sanity check*/
                printf("this grouple has %d words \n",pGrouple->iWords);
                status = VEOS_SUCCESS;
                }/*endif*/

             return (status);

             }/*somePrimName*/
```

The GroupleSpace Manager -- Nancy

Nancy is the groupleSpace manager, providing basic operations, creation, insertion, deletion, selection, and modification of grouples in the groupleSpace. All functions mentioned below are more fully documented in the nancy_doc_appendix. The VEOS is written in C, so indices start at 0, not 1. the first cell of a vector is vector[0], the second, vector[1]. The first word field of a grouple, pGrouple->pWordList[0] is its keyword. The phrase "named grouple" refers to a grouple with a given name as its first word.

The paths towards grouplefication

1.

If you merely need an empty grouple, use newGrouple(), which takes a handle to a grouple as an argument. The pointer (*handle) must be defined. That is, you can't define a handle, and expect this function to work (because the function dereferences the handle, trying to set up the fields of the struct/grouple). Define a pointer, and pass a reference to it.

THIS

```
        TPGrouple   pNewGrouple;
        status = newGrouple(&pGrouple);
```

NOT THIS

```
        THGrouple   hNewGrouple;
        status = newGrouple(hNewGrouple); /* will seg fault */
```

2.
    If you need a named new grouple (for instance, a data grouple with a known first word that you want to create at startup for other primatives to use), use setupDataGrouple(). Pass the text of the first word (the keyword).

```
            char  *sText = "DATA-Tutorial-Example"
            status = setupDataGrouple(sText);
```

3.
    If you want to put a mess of words into a new grouple, use slopDataIntoGrouple(). Remember to pass a handle, which must point to a real pointer (define a pointer and pass a reference to it).   Each word must be a string (null terminated char vector). If you want to put numbers in, you must first convert them into strings (sprintf is easy, or write an ftoa() function). Yes, this is a drag.

```
            TPGrouple   pGrouple;
            int        iHowManyWords = 3;
            float      fDummy = 2.4;
            char       sConvertedDummy[256];

            sprintf(sConvertedDummy,"%f",fDummy);
            status = slopDataIntoGrouple(&pGrouple,iHowManyWords,
                                  "use", "this", sConvertedDummy);
```

4.
    If you want to put a mess of words into a grouple and want to INSERT the grouple DIRECTLY into the groupleSpace, without ever using the grouple yourself, use postDataAsGrouple.

```
            int        iHowManyWords = 3;

            status = postDataAsGrouple(iHowManyWords,
                                  "send", "this", "away");
```

REMEMBER to free the space associated with grouples that local procedures may have created, by calling disposeGrouple(hGrouple). Don't pass a pointer, pass a handle. It is considered unfriendly to dispose of a grouple which was created elsewhere. Local procedures should clean up their own messes.

```
            status = disposeGrouple(&pLocalGrouple);
```

The paths toward grouple Insertion

While creating grouples is relatively simple, inserting grouples is made slightly more complex, because of the many nancy options available to control the relative positions of grouples in the space.

Nancy supports the notion of multiple clients, each of whom may be operating on a shared groupleSpace. Clients are known by their client ID, which is of type TPWhoInfo. Each shell has its own (hidden) client ID; it is up to the application programmer to request and use a unique client ID, so as not to conflict with the shell.

Nancy maintains a SELECTION for each client, which is one grouple. The current selection is set only through calls to querySpace(). Many nancy calls use this selection as a relative position point in the grouple space. Some, like killGrouple, use the selection directly.

To get a TPWhoInfo block, use the newWhoParamBlock() function. The second field of this function is a chunk of descriptive text which will be inserted into the TWhoInfo struct, for use in debugging nancy, should strangeness occur. Typically it is the name of the application.

!! It is useful to have your TPWhoParamBlock globally accessible. !!

```
TPWhoInfo       pApplicationID;

status = newWhoParamBlock(&pApplicationID, "myApplication");
```

Actual ways to insert grouples into the space

1.
Use postDataAsGrouple as described above.

2.
Use insertGrouple(pGrouple, nancyFlags, pCallerID).

The first field is the grouple, the second, one of the set of nancy insertion Flags, which are

```
<VEOS_Nancy_InsertBeforeSelect,
VEOS_Nancy_InsertAtTop,
VEOS_Nancy_Append>,
```

and the third parameter is the TPWhoParamBlock of the client requesting the insertion.

You may pass null for the callerID if you are not inserting before your selection (since nancy only cares who you are when you reference your selection). Note that nancy inserts a copy of the grouple you give it, your copy of the grouple is unchanged, and *YOU* must free its space explicitly.

```
            TPWhoInfo              pMyID;
            TPGrouple              pGrouple;

            status = newGrouple(&pGrouple);

            if (status == VEOS_SUCCESS)
                  status = newWhoParamBlock(&pMyID,"myApplication);

            if (status == VEOS_SUCCESS)
                status = insertGrouple( pGrouple,VEOS_Nancy_InsertAtTop,
                                       null);

            if (status == VEOS_SUCCESS)
                status = insertGrouple( pGrouple,
                                       VEOS_Nancy_InsertBeforeSelect,
                                       pMyID);

            if (status == VEOS_SUCCESS)
                    disposeGrouple(&pGrouple);
```

Modification of Grouples

It is possible to modify Grouples held locally and in the grouple space. changeDataGrouple modifies a named grouple already in the space, and setGroupleWord changes a locally held grouple. Both functions expect position/value pairs. In addition, one may completely replace one data grouple with another with replaceDataGrouple.

Deletion of words in a grouple is achieved by sending postition/NULL pairs of changes to changeDataGrouple or setGroupleWord. The position specified words are deleted, and all subsequent words are renumbered. So, if you want to delete several words, delete from the highest numbered words to the lowest numbered words, so as not to pick up errors from renumbering artifacts.

this code fragment deletes the last two word of pGrouple. Note also, that word position (indices) start at 0, NOT 1. Word 0 is the first word (the keyword), word n-1 is the Nth word.

```
            if ((pGrouple) && (pGrouple->iWords > 1)) {
                  status =  setGroupleWord(pGrouple,
                                   pGrouple->iWords, NULL);

                }/*endif sane*/
```

this code fragment deletes the 2nd word of a data-grouple called "DATA-My-Grouple". The second parameter is how many index/value pairs are following. The second word is at index 1 (start counting at 0).

```
            status = changeDataGrouple("DATA-My-Grouple",1,
                                1,NULL);
```

SUPPOSE you want to delete the last two words of a grouple in the dataSpace. How do you know what the indices of those words are?! You don't. Which brings us to a

Digression into Querying the GroupleSpace (and making a selection)

You may query the grouple space only by using querySpace(). Also, this is the only way to make a selection. This function has many many options, most of which i have never used. querySpace() looks like this:

```
            TVeosErr querySpace(char      *sMatchWord,
                                THGrouple   hGroupleCopy,
                                int         iSelectFlags,
                                int         iSearchFlags,
                                int         iSearchDirection,
                                TPWhoInfo   pCallerID)
```

sMatchWord is the text to match against the keyword (word 0) of grouples in the space. The wildcard character '*' may occur in the search string. "*" will match every grouple.

hGroupleCopy is a handle to a defined TPGrouple (define the TPGrouple and pass a reference to it.) Pass this if you want a copy of the grouple which is found. If you only want to know that the named grouple exists (you don't need a copy), pass NULL.

```
    iSelectFlag is one of <VEOS_Nancy_SelectResult,
                    VEOS_Nancy_DontChangeSelect>.
```

If you choose not to select, you may pass NULL for pCallerID, as nancy does not care who you are if you are not selecting.

```
    iSearchFlags is from the set <VEOS_Nancy_SearchFromSelect,
                        VEOS_Nancy_SearchFromTop,
                        VEOS_Nancy_SearchFromBottom >.
```

Once again, if you SearchFromSelect, pCallerID must not be NULL.

```
    iSearchDirection is one of <VEOS_Nancy_SearchBackward,
                        VEOS_Nancy_SearchForward>.
```

pCallerID is a TPWhoParamBlock, typically one you got from newWhoParamBlock(). You may pass NULL for this unless you are selecting the result, or searching from the current selection.

When you want to UNSELECT a grouple, use unselectGrouple(). It is considerate to unselect a grouple when you are done, so that some other client may select it. Pass your TPWhoParamBlock as parameter.

```
                status = unselectGrouple(pMyID)
```

Return to Sophisticated Grouple Modification, continued.

Back to the previous problem, how to delete the last two words of a grouple in the space. Here's one way:

First, query the space to find the grouple.
Then, look at the iWords field of the grouple to find out its size
Last, call changeDataGrouple with the correct indices, remembering that the iWords field starts counting at 1, but the indices start at 0.

```
                TPGrouple     pFoundGrouple;
                TVeosErr      status;

                status = querySpace("DATA-My-Grouple",
                          &pFoundGrouple,
                          VEOS_Nancy_DontChangeSelect,
                          VEOS_Nancy_SearchFromTop,
                          VEOS_Nancy_SearchForward,
                          NULL);
```

```
        if (status == VEOS_SUCCESS) {
          if (pFoundGrouple->iWords > 1) {
              status = changeDataGrouple("DATA-My-Grouple",2,
                          pFoundGrouple->iWords - 1,NULL,
                          pFoundGrouple->iWords - 2,NULL)
              }/*endif happy*/
          else
              status = VEOS_NEUTRAL; /* no error, no success */

          disposeGrouple(&pFoundGrouple);  /* throw copy away */
          }/*endif successful search*/
```

Note that you could also setGroupleWord the changes into your local copy, and then call replaceDataGrouple. Why you would want to do this, i don't know.

```
        status = setGroupleWord(pFoundGrouple,0,NULL);

        if (status == VEOS_SUCCESS)
          status = replaceDataGrouple("DATA-My-Grouple",
                                pMyID, pFoundGrouple);
```

Deleting grouples

i have always only deleted my selection. see the nancy_doc_appendix for more options.

```
    status =  killGrouple(pMyID, VEOS_Nancy_DeleteSelected);
```

Putting it together, stepping through the groupleSpace

Aha, you think, if i could only step through each grouple, one at a time, i could write my own shell. The groupleSpace is not circular, so querySpace returns an error when it has reached the bottom. Here's how such a loop might look.

```
TVeosErr    iQueryErr;
TVeosErr    iSuccess    = VEOS_SUCCESS;
TPGrouple   pNextGrouple;
/*TPWhoParamBlock pShellID  is a global */


/* first time through, start at the top of the space */

iQueryErr = querySpace("*",
                     &pNextGrouple,
                     VEOS_Nancy_SelectResult,
                     VEOS_Nancy_SearchFromTop,
                     VEOS_Nancy_SearchForward,
                     pShellID);

while ((iQueryErr == VEOS_SUCCESS) &&
         (iSuccess != VEOS_EXIT)) {


    /** process this grouple **/
    iSuccess = munchGrouple(pNextGrouple);


    /* done with local copy */
    disposeGrouple(&pNextGrouple);


    /** delete one-time function calls **/
    if (iSuccess == VEOS_SUCCESS)
         killGrouple(pShellID, VEOS_Nancy_DeleteSelected);


    /** find the next grouple and select it **/
    iQueryErr = querySpace("*",
                     &pNextGrouple,
                     VEOS_Nancy_SelectResult,
                     VEOS_Nancy_SearchFromSelect,
                     VEOS_Nancy_SearchForward,
                     pShellID);


    }/*while*/
```