Combinations Template B/combt

Purpose
Code to support the combination kind of value constructor.

§**1. Head.** As ever: if there is no heap, there are no combinations (in this sense).

```
#IFDEF MEMORY_HEAP_SIZE; ! Will exist if any use is made of heap
```

§**2. KOV Support.** See the "BlockValues.i6t" segment for the specification of the following routines.

```
[ COMBINATION_TY_Support task arg1 arg2 arg3;
    switch(task) {
        CREATE_KOVS:     arg3 = COMBINATION_TY_Create(arg2);
                         if (arg1) COMBINATION_TY_CopyRawArray(arg3, arg1, 2);
                         return arg3;
        CAST_KOVS:       rfalse;
        DESTROY_KOVS:    return COMBINATION_TY_Destroy(arg1);
        PRECOPY_KOVS:    return COMBINATION_TY_PreCopy(arg1, arg2);
        COPY_KOVS:       return COMBINATION_TY_Copy(arg1, arg2);
        COMPARE_KOVS:    return COMBINATION_TY_Compare(arg1, arg2);
        READ_FILE_KOVS:  rfalse;
        WRITE_FILE_KOVS: rfalse;
        HASH_KOVS:       return COMBINATION_TY_Hash(arg1);
    }
];
```

§**3. Creation.** A combination is like a list, but simpler; it has a fixed, usually short, size. On the other hand, its entries are not all of the same kind as each other.

Combinations are stored as a fixed-sized block of word entries. The first block is the only header information: a pointer to a further structure in memory, describing the kind. The subsequent blocks are the actual records. Thus, a triple (x, y, z) uses 4 words.

```
Constant COMBINATION_KIND_F = 0; ! A pointer to a block indicating the kind
Constant COMBINATION_ITEM_BASE = 1; ! List items begin at this entry
[ COMBINATION_TY_Create kind comb N i bk v;
    N = KindBaseArity(kind);
    comb = BlkAllocate((COMBINATION_ITEM_BASE+N)*WORDSIZE, COMBINATION_TY, BLK_FLAG_WORD);
    BlkValueWrite(comb, COMBINATION_KIND_F, kind);
    for (i=0; i<N; i++) {
        bk = KindBaseTerm(kind, i);
        if (KOVIsBlockValue(bk))
            v = BlkValueCreate(bk);
        else
            v = DefaultValueOfKOV(bk);
        BlkValueWrite(comb, COMBINATION_ITEM_BASE+i, v);
    }
    return comb;
];
```

§**4. Setting Up.** NI needs to compile code which will create constant combinations at run-time: the following routine is convenient for that. A "raw array" in this routine's sense is an array `raw` such that `raw-->2` contains the number of items, `raw-->1` the kind of value, and `raw-->3` onwards are the items themselves; note that this is the same format used for constant lists.

```
[ COMBINATION_TY_CopyRawArray comb raw rea len i ex bk v w;
    if ((comb==0) || (BlkType(comb) ~= COMBINATION_TY)) return false;
    ex = BlkValueExtent(comb);
    len = raw-->2;
    if ((len+COMBINATION_ITEM_BASE > ex) &&
        (BlkValueSetExtent(comb, len+COMBINATION_ITEM_BASE) == false)) return 0;
    BlkValueWrite(comb, LIST_LENGTH_F, len);
    if (rea == 2) bk = BlkValueRead(comb, COMBINATION_KIND_F);
    else {
        bk = raw-->1;
        BlkValueWrite(comb, COMBINATION_KIND_F, bk);
    }
    for (i=0:i<len:i++) {
        v = raw-->(i+3);
        if (KOVIsBlockValue(bk)) v = BlkValueCreate(bk, v);
        BlkValueWrite(comb, i+COMBINATION_ITEM_BASE, v);
    }
    #ifdef SHOW_ALLOCATIONS;
    print "Copied raw array to comb: "; COMBINATION_TY_Say(comb, 1); print "^";
    #endif;
    return comb;
];
```

§**5. Destruction.** If the comb items are themselves block-values, they must all be freed before the comb itself can be freed.

```
[ COMBINATION_TY_Destroy comb kind no_items i bk;
    kind = BlkValueRead(comb, COMBINATION_KIND_F);
    no_items = KindBaseArity(kind);
    for (i=0; i<no_items; i++) {
        bk = KindBaseTerm(kind, i);
        if (KOVIsBlockValue(bk))
            BlkFree(BlkValueRead(comb, i+COMBINATION_ITEM_BASE));
    }
    return comb;
];
```

§**6. Copying.** Again, if the comb contains block-values then they must be duplicated rather than bitwise copied as pointers.

```
Global precopied_comb_kov;
[ COMBINATION_TY_PreCopy lto lfrom comb no_items i nv bk;
    precopied_comb_kov = BlkValueRead(lto, COMBINATION_KIND_F);
];
[ COMBINATION_TY_Copy lto lfrom no_items i nv kind bk;
    kind = BlkValueRead(lto, COMBINATION_KIND_F);
    no_items = KindBaseArity(kind);
    BlkValueWrite(lto, COMBINATION_KIND_F, precopied_comb_kov);
    for (i=0; i<no_items; i++) {
        bk = KindBaseTerm(kind, i);
        if (KOVIsBlockValue(bk)) {
            nv = BlkValueCreate(bk);
            BlkValueCopy(nv, BlkValueRead(lfrom, i+COMBINATION_ITEM_BASE));
            BlkValueWrite(lto, i+COMBINATION_ITEM_BASE, nv);
        }
    }
];
```

§**7. Comparison.** This is a lexicographic comparison and assumes both combinations have the same kind.

```
[ COMBINATION_TY_Compare listleft listright delta no_items i cf kind bk;
    kind = BlkValueRead(listleft, COMBINATION_KIND_F);
    no_items = KindBaseArity(kind);
    for (i=0; i<no_items; i++) {
        bk = KindBaseTerm(kind, i);
        cf = KOVComparisonFunction(bk);
        if (cf == 0 or UnsignedCompare) {
            delta = BlkValueRead(listleft, i+COMBINATION_ITEM_BASE) -
                BlkValueRead(listright, i+COMBINATION_ITEM_BASE);
            if (delta) return delta;
        } else {
            delta = cf(BlkValueRead(listleft, i+COMBINATION_ITEM_BASE),
                BlkValueRead(listright, i+COMBINATION_ITEM_BASE));
            if (delta) return delta;
        }
    }
    return 0;
];
[ COMBINATION_TY_Distinguish txb1 txb2;
    if (COMBINATION_TY_Compare(txb1, txb2) == 0) rfalse;
    rtrue;
];
```

§8. Hashing.

```
[ COMBINATION_TY_Hash comb  kind rv no_items i bk;
    rv = 0;
    kind = BlkValueRead(comb, COMBINATION_KIND_F);
    no_items = KindBaseArity(kind);
    for (i=0: i<no_items: i++) {
        bk = KindBaseTerm(kind, i);
        rv = rv * 33 + KOVHashValue(bk, BlkValueRead(comb, i+COMBINATION_ITEM_BASE));
    }
    return rv;
];
```

§9. Printing.

```
[ COMBINATION_TY_Say comb format no_items v i kind bk;
    if ((comb==0) || (BlkType(comb) ~= COMBINATION_TY)) return;
    kind = BlkValueRead(comb, COMBINATION_KIND_F);
    no_items = KindBaseArity(kind);
    print "(";
    for (i=0; i<no_items; i++) {
        if (i>0) print ", ";
        bk = KindBaseTerm(kind, i);
        v = BlkValueRead(comb, i+COMBINATION_ITEM_BASE);
        if (bk == LIST_OF_TY) LIST_OF_TY_Say(v, 1);
        else PrintKindValuePair(bk, v);
    }
    print ")";
];
#IFNOT; ! IFDEF MEMORY_HEAP_SIZE
[ COMBINATION_TY_Support t a b c; rfalse; ];
[ COMBINATION_TY_Say comb; ];
#ENDIF; ! IFDEF MEMORY_HEAP_SIZE
```