Ownership

Objects ownership system

Red's objects ownership system is an extension of object's event support introduced in previous releases. Now, an object can own series it references, even nested ones. When an owned series is changed, the owner object is notified and its `on-deep-change*` function will be called if available, allowing the object to react appropriately to any change.

The prototype for on-deep-change* is:

```
on-deep-change*: func [owner word target action new index part][…]
```

The arguments are:

- owner: object receiving the event (object!)
- word: object's word referring to the changed series or nested series (word!)
- target: the changed series (any-series!)
- action: name of the action applied (word!)
- new: new value added to the series (any-type!)
- index: position at which the series is modified (integer!)
- part: number of elements changes in the series (integer!)

Action name can be any of: `random, clear, cleared, poke, remove, removed, reverse, sort, insert, take, taken, swap, trim`. For actions "destroying" values, two events are generated, one before the "destruction", one after (hence the presence of `cleared, removed, taken` words).

When modifications affect several non-contiguous or all elements, `index` will be set to -1. When modifications affect an undetermined number of elements, `part` will be set to -1.

Ownership is set automatically on object creation if `on-deep-change*` is defined, all referenced series (including nested ones), will then become owned by the object. The `modify` action has been also implemented to allow setting/clearing ownership post-creation-time.

As for `on-change, on-deep-change*` is kept hidden when using `mold` on an object. It is only revealed by `mold/all`.

Here is a simple usage example of object ownership. The code below will create a numbers object containing an empty list. You can append only integers to that list, if you fail to do so, a message will be displayed and the invalid element removed from the list. Moreover, the list is always sorted, wherever you insert or poke a new value:

```
    numbers: object [
        list: []

        on-deep-change*: function [owner word target action new index part][
            if all [word = 'list find [poke insert] action][
```

```
                forall target [
                    unless integer? target/1 [
                        print ["Error: Item" mold target/1 "is invalid!"]
                        remove target
                        target: back target
                    ]
                ]
                sort list
            ]
        ]
    ]

    red>> append numbers/list 3
    == [3]
    red>> insert numbers/list 7
    == [3 7]
    red>> append numbers/list 1
    == [1 3 7]
    red>> insert next numbers/list 8
    == [1 3 7 8]
    red>> append numbers/list 4
    == [1 3 4 7 8]
    red>> append numbers/list "hello"
    Error: Item "hello" is invalid!
    == [1 3 4 7 8]
    red>> numbers
    == make object! [
        list: [1 3 4 7 8]
    ]
```

Object ownership is deeply used in Red/View, in order to achieve the binding between face objects and the widgets on screen, and the automatic synchronization without the need to call `show`.

The work on this is not yet completed, more object events will be provided in future releases and the ownership support extended to enable objects to own more datatypes. More documentation will be provided once the work on that is finished. In the future, its use will be extending to other frameworks and interfaces. Such "reactive objects" will be called "live objects" in Red's jargon.