mergJSON

mergJSON is a JSON encoding/decoding external supporting all the platforms LiveCode supports.

License

This external is available under a dual-license. If you are doing commercial work with it please purchase a commercial license from mergExt otherwise you are free to use under the GPL 3 license. The intention is that it's free to use wherever LiveCode is free to use.

Installation

Look here for installation instructions for the IDE and desktop standalones.

Look here for installation instructions for iOS and Android.

To install in server then you just need to drop the external into the externals folder. For OS X use mergJSON.dylib rather than mergJSON.bundle on server.

Design

This external has been implemented to encode/decode UTF8 JSON to a LiveCode array as quickly as possible. Currently externals can not work with multi-dimensional LiveCode arrays so the intention is it is used with two functions like this:

```
-- pArray - array to be encoded
-- pForceRootType - can force the root to be an object if it looks like an ar
-- pPretty - include whitespace
function ArrayToJSON pArray,pForceRootType,pPretty
   repeat for each key tKey in pArray
      if pArray[tKey] is an array then
         put "}"&ArrayToJSON(pArray[tKey]) into pArray[tKey]
```

```
      end if
   end repeat
   return(mergJSONEncode("pArray",pForceRootType,pPretty))
end ArrayToJSON

function JSONToArray pJSON
   local tArray,tKeys
   repeat for each line tKey in mergJSONDecode(pJSON,"tArray")
      put JSONToArray(tArray[tKey]) into tArray[tKey]
   end repeat
   return tArray
end JSONToArray
```

For most cases the above is all the documentaion you will need. There are however some quirks to the external you might need to know about.

QUIRK 1

If the external encounters } as the first char of an array element it will assume the string is already encoded. This is how the recursive function is able to translate a multi-dimensional array into JSON. It's also the only way to get an empty array or object into your JSON by setting the element value to }[] or }{}.

QUIRK 2

If the keys of the array are numbered 1...N where N is the number of elements then the array is encoded as a JSON array. If not it's encoded as a JSON object. It's possible that in some cases you want to force what looks like an array to be an object. If that's the case then use the pForceRootType parameter with a value of "object".

QUIRK 3

Sometimes strings look like numbers so if you need to force a value to be a string then use the pForceRootType parameter with a value of "string". This only works when you are encoding a normal variable rather than an array so you need to pre-parse your array to pre-encode any strings that may look like numbers.

QUIRK 4

If the value looks like a real number it is encoded as an IEEE 754 double. In order to represent the number uniquely a double may need a precision of 15 to 17 decimal places. What this means is that any encoded then decoded doubles may need to be formatted or rounded to the precision you require before displaying it to a user because something like 0.657679 when converted to a double will become 0.65767900000000001. I'm expecting the people at IEEE are smarter than me so I'm not going to mess with that...

> *If a decimal string with at most 15 significant decimal is converted to IEEE 754 double precision and then converted back to the same number of significant decimal, then the final string should match the original; and if an IEEE 754 double precision is converted to a decimal string with at least 17 significant decimal and then converted back to double, then the final number must match the original.*
