Data Collections

Welcome to the Fourth Dimension
(and beyond)

Martin Phillips
Ladybridge Systems Ltd

International Spectrum Conference, 2014

Multivalue – Are we at its limits?

We all understand the power of the multivalue data model

Discarding the First Rule of Normalisation gives us simpler, faster and more maintainable applications than relational databases

Today’s data may extend beyond the three dimensions that multivalue provides

XML and JSON may go much further.

JSON – JavaScript Object Notation

JSON defines data as a character string holding name/value pairs

JSON allows unlimited nesting of objects.

Multivalue has excellent string processing

It is not hard to write a JSON parser or update tool

Achieving good performance may be much harder.

JSON – JavaScript Object Notation

We need to separate JSON as a data representation for transmission or storage from the need for efficient processing of arbitrarily multi-dimensional data in an application

JSON is an external representation of our data

Technology moves forward. JSON may be replaced by some new format in the future.

Data Collections

QM supports Data Collections as arbitrarily multi-dimensional sets of name/value pairs

Perhaps other multivalue vendors will follow

Any apparent similarity in some concepts to MongoDB is coincidental.

Data Collections - Data Types

Multivalue Basic uses type variant data items

JSON objects hold values with distinct data types

- Strings
- Numbers
- Booleans (true, false)
- Null

Data items can be grouped into

- Single dimensional arrays
- Nested objects

Data collections must support all of this (or more).

Data Collections

A data item within a data collection may be of any QM data type

Some types are probably irrelevant but all are supported

Because a data collection is itself a QM data type, data collections can be nested to any depth.

Parsing JSON into a Data Collection

The JPARSE() function creates a data collection from a JSON string

```
VAR = JPARSE(JSON.STRING)
```

Parsing a realistic complex JSON string with 2600 items takes about 650µS on a 2.7GHz PC.

Accessing Data in a Data Collection

Referencing a value in a data collection is very similar to referencing a dynamic array

Instead of numeric positions (field, value, subvalue), we use element names

```
ITEM = VAR{name}
```

We can use a quoted constant, a variable or an expression for the name.

Accessing Data in a Data Collection

To access a value in a nested object, we need two or more names

ITEM = VAR{*name1*, *name2*}

This is no different from accessing a value position in a dynamic array

Because a nested object is just a data item, we could do this as

ITEM = VAR{*name1*} {*name2*}

Element Paths

We can also use an *element path* in which the names are separated by forward slashes

ADDR = VAR{“client/address”}

This is useful when the structure of the data is not fixed

The different syntaxes can be mixed.

Building a JSON String

The JBUILD() function builds a JSON string from a data collection

```
JSON.STRING = JBUILD(VAR)
```

It is important that parsing and rebuilding a JSON string should produce a result that is equivalent to the original data.

Numeric Values

JSON provides several formats for numeric data:

```
123
123.00
1.23E2
12300E-2
```

In QM, all of these will be held in memory as integer value 123

Floating point values are used where necessary

Building a JSON string will produce equivalent values but perhaps not the same format.

Boolean Values

Multivalue uses 1 and 0 for True and False

It is not acceptable that parsing a JSON true value and then rebuilding the JSON string replaces it with a numeric 1

Since release 3.2-2 (October 2013), QM has supported an internal Boolean data type

This is completely compatible with use of numeric values, converting as needed.

The Null Value

QM release 3.2-2 also added the SQL style null value data type

Values can be set or tested as null but dynamic operations on null items is not currently supported.

Working with Data Collections

An empty collection is created with

```
CLIENT = COLLECTION()
```

Or another collection is copied with

```
CLIENT = COLLECTION(OTHER.CLIENT)
```

Copying a collection variable gives a second reference to the same collection, not a copy

```
CLIENT = OTHER.CLIENT
```

This is similar to file variables and some others.

Working with Data Collections

The member names in a collection can be determined using ENUMERATE()

```
NAMES = ENUMERATE(VAR)
```

Nested collections must be enumerated separately.

Modifying Data Collections

Members of a collection can be added or updated using

```
CLIENT{'name'} = CLIENT.NAME
```

Alternatively, a modified form of the INS statement can be used

```
INS CLIENT.NAME AS VAR{'name'}
```

Modifying Data Collections

Intermediate levels are inserted automatically (just like dynamic arrays)

```
CLIENT = COLLECTION()
CLIENT{‘address/zip’} = ‘96439’
```

Modifying Data Collections

Members are deleted using the DEL statement.

```
DEL CLIENT{‘phone’}
```

Arrays in Data Collections

Collections can include single dimensional arrays

An empty array is created with

CLIENT{‘contacts’} = MAT()

A standard dimensioned array can be copied to a collection

**CLIENT{‘contacts’} =
MAT(CONTACT.NAMES)**

Arrays in Data Collections

Array elements are referenced by using their element number as the name

NAME = CLIENT{‘contacts/1’}

or

NAME = CLIENT{‘contacts’, 1}

Arrays automatically resize to fit the content

INS and DEL can be used to insert or delete array elements.

Arrays in Data Collections

An element path may contain an asterisk as the member name within an array to return a multivalued list of element values

```
PRODUCTS = ORDER{‘detail/*/prod.no’}
```

Use of a second asterisk returns a subvalue list

```
SERIAL = ORDER{‘detail/*/serial.no/*’}
```

Data Collection Files

A data collection can be stored in JSON form in a standard data file

The COLLECTION option of CREATE.FILE creates a file in which the records are data collections stored in an optimised format

Reading a record from the file creates a collection variable

Data written to the file must be a collection variable.

Data Collection Files

The dictionary of a collection file may only use a field number (0) to reference the record id

Collection members are referenced using E-type (element) dictionary items

These are like D-type but have the element path in field 2

The element path may use the asterisk syntax to return multivalued data.

I-type dictionary items are supported, including TRANS() to fetch data from another collection file.

Data Collection Files

Data collection files may use:

•Indices

•Record level encryption

•Replication

•Transactions

•Triggers

Linked Data Collection Files

A string value in a collection can be used to reference another collection record

> ***"file:id"***

or

> ***":id"***

The EXPAND() function follows the link and updates the data collection in memory to include the linked item.

> **OK = EXPAND(VAR{*link.path*})**
> **OK = EXPAND(VAR{*link.path*}, *file.var*)**

Data Collection Files in Queries

The query processor can create reports based on data from data collection files by use of E-type dictionary items

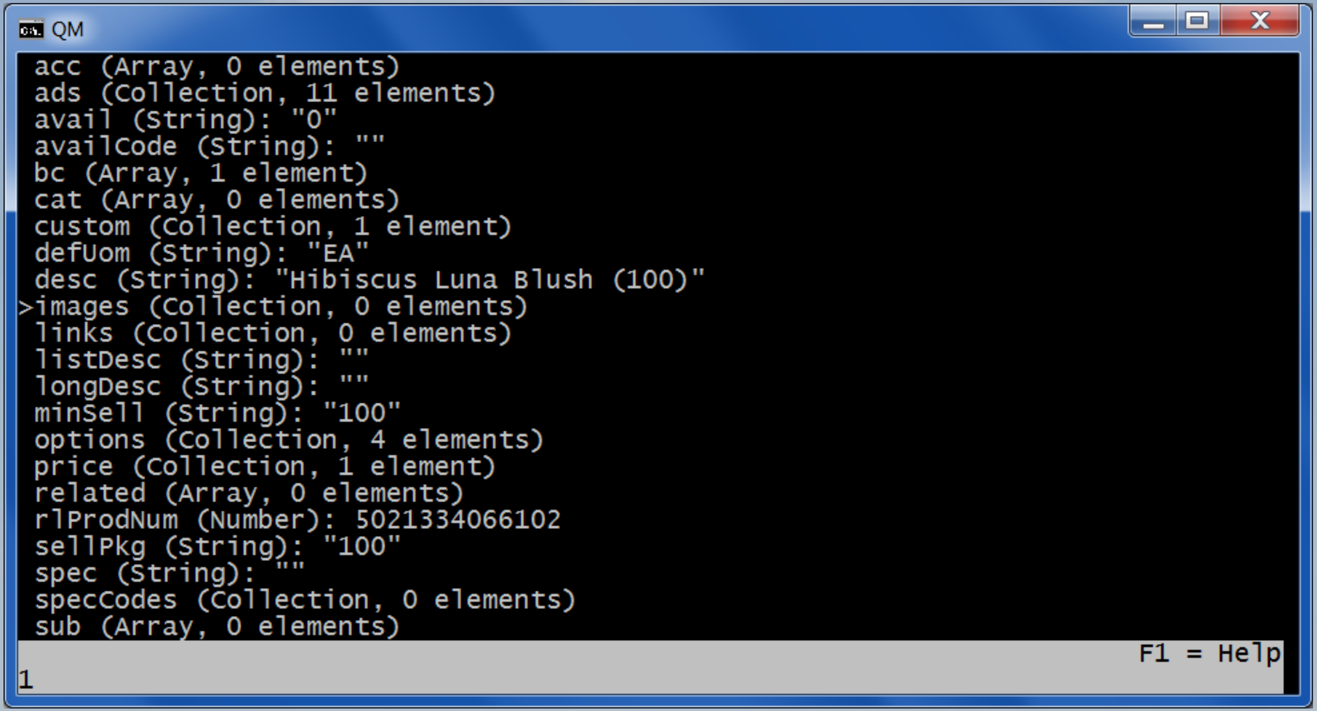
The ELEMENT keyword can be used to reference an item by element path if there is no dictionary item

```
LIST CLIENTS ELEMENT ‘address/zip’
```

The Data Collection Editor

Use as a command or as a subroutine

Allows viewing and editing of collections.



```
 acc (Array, 0 elements)
 ads (Collection, 11 elements)
 avail (String): "0"
 availCode (String): ""
 bc (Array, 1 element)
 cat (Array, 0 elements)
 custom (Collection, 1 element)
 defUom (String): "EA"
 desc (String): "Hibiscus Luna Blush (100)"
>images (Collection, 0 elements)
 links (Collection, 0 elements)
 listDesc (String): ""
 longDesc (String): ""
 minSell (String): "100"
 options (Collection, 4 elements)
 price (Collection, 1 element)
 related (Array, 0 elements)
 rlProdNum (Number): 5021334066102
 sellPkg (String): "100"
 spec (String): ""
 specCodes (Collection, 0 elements)
 sub (Array, 0 elements)
                                                                      F1 = Help
1
```

Summary

Data collections allow us to process data that goes beyond the three dimensions supported by the multivalue model

The concepts that an application developer must learn are very close to the dynamic array operations that they already use.

QUESTIONS?

Open QM

Ladybridge Systems

taking multivalue where it has never been before …