

Beyond4P

Beyond former expectations of
Performance,
Productivity,
Predictability and
Professionalism

Turning Big Data to Smart Data

An unparalleled programming language for
high performance data and table processing

By Georg A. zur Bonsen

Release **7.04** 07.04.2020

A Release on Independence Day

- Another performance boost particularly on large and very large tables and data structures
- Extended locale support
- Console output: Supports text and background colors and moving cursor position
- Smart tokenizer function provided to parse text for cleanup purposes
- Overhaul of 'literal' function and loading HTML tables
- Runs embedded B4P code in github markdown documents
- Increased support for scientific notation in tables

More than 780 functions, incl. 200 table manipulation functions

All rights reserved, copyright © 2012...2020 by Georg zur Bonsen

Cover photo: Office building with colorful window shades located in Schlieren near Zürich, Switzerland.

Key Versions	3.00 – Introducing Beyond4P as Script Language (Earlier versions were C++ libraries)
	4.00 – Unicode
	5.00 – Formatting and Style introduced
	6.00 – Unleashed 64 bit performance
	7.00 – Runs on both LINUX and WINDOWS platforms

Release Notes	7.04 – Various Improvements and Clean-ups
Bugfixes	<p>Loading JSON files: Crashed when encountering invalid literals. Double quotation marks inside strings to be used as part of strings were misinterpreted as ending strings. Both issues have been resolved.</p> <p>JSON: Previous version forgot about reading / writing empty arrays as they are different from <i>null</i>.</p> <p>JSON: Beyond4P now checks for repeated use of same name within one object and issues error messages accordingly. Previously, later assignments to the same name simply overwrote the existing assignment.</p> <p>In case of exceptions (e.g. syntax errors, functions returning errors), local variables are no longer for inspection while interactive mode applies. This has been fixed.</p> <p>Bugfixed and overhauled 'literal' and 'table load(... HTML)' functions</p>
New general features	<p>If activated, scientific notation (exponents) is supported in accessing numbers in tables and input (...) function.</p> <p>Automatic array creation. E.g. if variable <code>a[]</code> is not yet existing or just a simple variable, then <code>a[3] = 1</code> would create an array with 4 elements, assign 1 to the last one and initialize the remaining ones as void. <code>a[5]</code> would extend that array, i.e. initialize <code>a[4]</code> as void value and <code>a[5]</code> with the next value provided.</p> <p>Github markdowns: Files ending with ".md" will recognize code blocks (delimited with `` symbols) and execute only those.</p>
New functions	<p>table append on same field (...) appends text contents to the last field in the last row.</p> <p>text color (...) and background color (...) changes colors on console</p> <p>cursor (...) sets cursor position on console</p> <p>tokenize (...) breaks a string down into tokens according to specified rules</p> <p>table lookup smart once (...) and table lookup smart once ignore case (...) combines advantages of both table lookup smart and table lookup once function families.</p>
Enhanced functionality	<p>table save (...) supports new format: TEXT. This assures original contents in tables saved as text without tampering with quotation marks.</p> <p>replace ... (...) and substitute ... (...) now allow replacing multiple old patterns by new patterns</p> <p>input (...): Numeric input now supports scientific notation</p>
Performance enhancements	Memory allocation and deallocation performance has been strategized which results to higher performance when processing very large data.
Obsolescence	

1	Introduction	8
1.1	Distinct Language Features	9
1.2	Licensing and Demo Mode	10
1.3	Installing Beyond4P	10
1.4	Starting Beyond4P	10
1.4.1	Windows	10
1.4.2	Linux	10
1.4.3	Command Line Options (switches)	10
2	Text, File Format and I/O Conventions	11
2.1	Directory and File Names	11
2.2	Definitions	11
2.3	UNICODE Character Set	11
2.3.1	Loading Files	11
2.3.2	Saving Files	13
2.4	CSV File Format	13
2.5	Console Output	13
2.6	Console Input	14
2.7	HTML Entity References	14
2.7.1	Program code interpretation	14
2.7.2	Reserved symbols	15
2.7.3	Conversion to Upper Case and Lower Case	15
3	Language Basics	16
3.1	Overall Program Syntax	16
3.2	Comments	16
3.2.1	Github markdown files	16
3.3	Basic Data Types	17
3.3.1	Literals	18
3.3.2	Numerals	20
3.3.3	Dates	21
3.3.4	Booleans	25
3.3.5	Voids	25
3.4	Parameter Sets	25
3.4.1	Basics	25
3.4.2	Ranges in Parameter Sets	26
3.4.3	Parameter Set Element Duplication	28
3.5	Picking Details	28
3.5.1	Read Access Case	28
3.5.2	Write Access Case	29
4	Variables	30
4.1	Introduction	30
4.2	Syntax	30
4.3	Variable Storage Model and Scopes (System, Global and Local Variables)	31
4.4	Defining Global Variables outside the main program	32
4.5	Structures	32
4.6	Default Member Variables	33
4.7	Referencing Structures with Numbers	34
4.8	Arrays	35
4.9	Indirect Variable References	35
4.10	Dynamic Referencing of Variable Members	36
4.11	Referencing Variables: Summary	36
4.12	References to Variables	37
4.12.1	Defining and Accessing Simple References to Variables	37
4.12.2	Releasing References to Variables	38
4.12.3	References in Member Variables	39

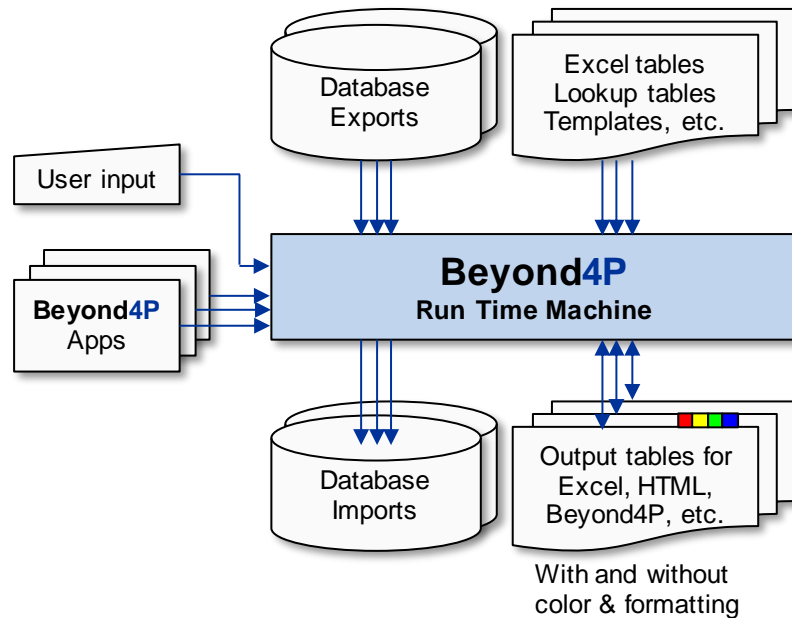
4.12.4	Invalid Use	39
4.12.5	Summary.....	40
4.13	Roundup on Variables.....	41
4.14	Protecting Variables.....	42
5	Tables	43
5.1	Creating Tables	43
5.2	Accessing Tables	44
5.3	Syntax	44
5.3.1	Full Table Specification – Simple Access	45
5.3.2	Full Table Specification – Lookup Access	46
5.3.3	Full Table Specification – Summary	47
5.3.4	Partial Table Specification – Simple Access	48
5.3.5	Partial Table Specification – Lookup Access	49
5.3.6	Partial Table Specification – Summary	49
5.4	Iterators	49
5.5	Automatic data conversions	49
5.6	Rules on Column Header Names	50
5.7	Referencing Table Cells – Special Rules	50
5.8	Memorizing Table Column Numbers (Secretly) for Performance	53
5.9	Horizontal Access (Multiple Cells in a Row).....	55
5.10	Vertical Access (Multiple Cells in a Column).....	57
5.11	Matrix Access (Multiple Rows and Columns).....	59
5.12	Special Cases.....	60
5.13	Partial Table Specifications – Next Level Up	61
5.14	Formatting Attributes	62
6	Formulas and Operators	64
6.1	Formula Basics	64
6.2	Logical Binary Operators.....	64
6.3	Comparison Operators.....	65
6.3.1	Wildcards.....	70
6.3.2	Ranges and Selections	71
6.3.3	Comparison Modifier Symbols.....	71
6.4	Comparison Expressions	72
6.5	Arithmetic Operators	73
6.5.1	Arithmetic operations with Dates (Time of day).....	75
6.6	Unary Operators	77
6.7	Parentheses	78
6.8	Assignment Operators	78
6.9	Ad hoc Operations	79
6.9.1	Basics	79
6.9.2	Restrictions	79
6.9.3	Ad hoc Operators.....	79
6.9.4	Target Objects for ad hoc operations	80
6.9.5	Defining the right moment to do ad hoc operation	80
6.9.6	Advanced Ad hoc operations.....	81
6.9.7	Summary of Ad Hoc Operations	82
6.10	Deep Operators	85
6.11	Type Conversions.....	87
7	Transactions.....	88
7.1	Transactions Basics	88
7.2	Transactions between Variables.....	88
7.3	Transactions from Tables to Variables.....	90
7.4	Transactions from Variables to Tables.....	92
7.5	Transactions between Tables	94
7.6	Transaction Assignment Operators	95

7.6.1	No Assignment Operator	95
7.6.2	The & (And) Assignment Operator	96
7.6.3	The (Or) Assignment Operator	97
7.6.4	The + (Plus) Assignment Operator	98
7.6.5	Transaction Assignment Operators: Examples	99
7.7	Parameter Sets, Arrays, Structures, Tables: Summary	100
8	Special Features	101
8.1	Sorting and Ranking	101
8.2	Exceptions	102
8.3	JavaScript Open Notation (JSON) Data Format	103
9	Functions and Procedures	104
9.1	Function Parameter Passing	104
9.2	Function Parameter Type Checking	104
9.3	Special Parameter Types: Table Columns Specifications	105
9.3.1	Shifted Column Specifications	106
9.4	Special Parameter Types: Table Rows Specifications	107
9.5	Return Values	108
9.6	Indirect Parameter Passing to Functions	108
9.6.1	Code Pieces passed as Parameters to Functions	109
10	Standard Function and Procedure Library	110
10.1	Flow Control Functions	110
10.2	Conditional Branches	110
10.2.1	Loops	111
10.2.2	Break and Continue	116
10.2.3	Running other Programs	117
10.2.4	Null Function	118
10.3	User Function Definition	119
10.4	Mathematics Functions	123
10.4.1	Check Numbers	123
10.4.2	Scalar Functions	124
10.4.3	Transcendental and Trigonometric Functions	125
10.4.4	Vector Functions	126
10.4.5	Matrix Manipulations	127
10.4.6	Matrix Functions	129
10.4.7	Select Minimum and Maximum Functions	130
10.4.8	Statistics Functions	132
10.4.9	Conditional Combination Functions	135
10.4.10	Finance and Business Functions	137
10.4.11	Interpolation Functions	144
10.5	Type Conversion Functions	146
10.6	Date and Time Functions	159
10.6.1	Date Functions	159
10.6.2	Time Functions	163
10.6.3	Stopwatch Functions	164
10.7	String Functions	165
10.7.1	Basic String Functions	165
10.7.2	Extract Part of Strings	166
10.7.3	Find and Replace	169
10.7.4	String Manipulations	172
10.7.5	Character Coding and Decoding	173
10.8	Parameter Set Functions	174
10.8.1	Basic Parameter Set Functions	174
10.8.2	Extract Parts of Parameter Sets	175
10.8.3	Find and Replace	178
10.9	File and Directory Functions	180
10.9.1	Basic File and Directory Functions	180
10.9.2	Directory Listing Functions	182
10.9.3	Deleting Files and Directories	184

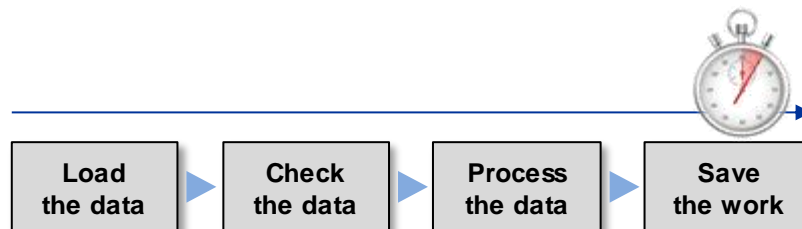
10.9.4	Renaming and Copying Files and Directories	186
10.9.5	Symbolic Links to Files and Directories	188
10.9.6	Internet Access Functions	189
10.10	System Functions	190
10.10.1	General System Functions	190
10.10.2	Licensing Functions and Query Privileges	190
10.10.3	Registry Functions (Windows only)	192
10.11	Table Functions	198
10.11.1	Creating, Initializing and Deleting Tables	198
10.11.2	Loading and Saving	203
10.11.3	Accessing Table Contents	208
10.11.4	Table Information and Verification	211
10.11.5	Searching Tables	224
10.11.6	Extracting Tables	228
10.11.7	Manipulating Table Columns	232
10.11.8	Manipulating Table Rows	236
10.11.9	Manipulating Table Cells	241
10.11.10	Manipulating Table Contents	242
10.11.11	Pivot Functions	250
10.11.12	Copying, Splitting and Renaming Tables	260
10.11.13	Combining Multiple Tables	262
10.11.14	Miscellaneous Table Functions	291
10.12	Comparison and Selection Functions	294
10.13	Variables Functions	299
10.13.1	Variable Information Functions	299
10.13.2	Structures and Arrays	300
10.13.3	Deleting Variables	305
10.13.4	Loading and Saving Variables	306
10.13.5	Protecting Variables	309
10.13.6	Global Variables	310
10.13.7	References to Variables	310
10.14	Resident Attributes Functions	311
10.15	Code Manipulation and Parsing Functions	312
10.16	Exception Functions	316
10.17	Input / Output Functions	320
10.17.1	Basic Input / Output Functions	320
10.17.2	Console Features	322
10.17.3	Sleeping and Waiting	323
10.18	Hash Functions	324
10.19	Debugging and Interactive Functions	325
11	Supplementary Function and Procedure Libraries	327
11.1	Style and Formatting Functions	327
11.1.1	Introduction	327
11.1.2	Defining User specific Colors	328
11.1.3	Add Style and Formatting	331
11.1.4	Translate Generic Type to Specific File Format	336
11.1.5	Reset Style and Formatting	337
11.2	File Compression and Decompression	338
11.3	Accessing Microsoft Open Office Document Files	339
11.4	Utility Library	342
12	Compile-Time Directives	343
13	Interactive Mode	345
14	Appendix: Programming Language Symbols	346
15	Language Syntax Summary	349
16	Reserved System Variables	352
17	Deprecated Features	355
18	Locales	356

1 Introduction

The Beyond4P tool supports a dedicated functional programming approach which provides high flexibility in processing tables generated by common database systems (e.g. SAP, Salesforce, Filemaker, Oracle which generate exports in HTML and MHTML file formats) as well as Excel (.xlsx, .xslm) and comma separated tables (.CSV files). Additional file formats will be added in the future when necessary. The key strength of Beyond4P is the ability to process contents in very large tables very effectively and generate simple as well as formatted outputs ready for analysis and visualization with Excel, web browsers and other tools.



Beyond4P provides an interpreted language which provides very powerful syntax structure, highly flexible data structures without need of initial declarations and a large function library. Small apps containing very few statements can already perform sophisticated operations such as merging two tables with different arrangements and formats.



1.1 Distinct Language Features

The following key features makes the Beyond4P language distinct from other popular programming languages:

- **The language is interpreted.**
 - The run-time machine is designed to interpret and process the code highly efficiently.
- **Names of variables, functions and tables may consist of multiple words and special characters.**

This allows you to use typical column headers, e.g. "Total Costs", without modifying them or putting quotations around. A valid function call could be " **table create** (customer table, supplier table, ...); "

Function names like "table process selected rows" is easier to read than "TableProcessSelectedRows".
- **Variables**
 - All simple variables end with brackets [], e.g. **index[]**
 - You can build structured variables simply by assigning values to them.
e.g. **animal[dog] = poodle; animal[dog, leg count] = 4;**
- **Values**
 - Texts do not need to be put into quotation marks unless you want to specify exact number of spaces between contents. **a[] = Bull dog; b[] = " Bull Dog ";**
- **UNICODE characters**

Regardless if running under Windows or LINUX, every character counts as 1 character, e.g. E, é, €, emojis, etc. This assures high degree of application compatibility across platforms
- **Basic Variable Types**
 - Numeral, Literal, Boolean, Date (with and without time), Parameter Set (and Void).
- **You can assign simple values (numbers, text, etc.) or parameter sets containing several of them**

children [Meyer] = Jane; children [Miller] = { Nicole, Jack, Timothy };
Nested parameter sets are supported and allow powerful features such as vector and matrix calculations.
- **The language uses *dynamic weak typing*.**

The first time you assign a variable, the type will be defined, and the type may change depending what you assign afterwards. You can start assigning a piece of text (literal), and later a number (numeral). Example: **a[] = 1; a[] = Hello;**
- **Tables**
 - All tables are in memory and therefore highly performant
 - Typcally referred as [table name: column, row].
 - In dedicated loops working tables, it is sufficient to specify the column name or number only, e.g. [Name]
Example: **table process(membership list, echo([Family Name], " ", [First Name]);**
- **You can also access multiple table cells** (e.g. part of or whole row, part of or whole column) with simple accesses, e.g. [geography table : { City, State, Country }, 3..4] retrieves values from row 3 and 4 in form of parameter sets, e.g. { { San Francisco, California, United States }, { Montréal, Quebec, Canada } }.
- The language provides an extensive function library starting from basic math (e.g. **min()**) as well as powerful table processing functions, e.g. **table merge ()** .
- **Transactions:** You can copy or combine variable structures with different ways of combining destination data. Example: **a[] + <== b[];**
- Various functions can take a variable (not fixed) number of parameters being passed.
- **You can also define your own functions with fixed or variable number of parameters.**

E.g. **define function (function name, { parameters ... }) { statements }**
- **Various functions support *indirect parameter passing*:** Instead of passing each item separately into the function, you can put them into a parameter set as described before and pass them in. Example:
a[] = { 2,3,4 }; echo(min(2,3,4)); echo(min(a[]); // Output is 2 in both cases.
- **Various functions take code pieces as parameters, e.g. expressions and statements.**
 - these parameters are executed multiple times inside the fuction.
 - Example: **table process(table name, echo([Name]));**
The 2nd parameter is a statement and executed for every row
- **The language provides deep operators** in order to process parameter contents directly, e.g.
a[] = { 1,2,3 } + { 4,5,6 }; b[] = { 1,2,3 } + ^ { 4,5,6 }; // a[] contains {1,2,3,4,5,6} and b[] contains {5,7,9}.
- **Control flow statements are function names**

Example: **if, while, for, for all table rows, switch, case, etc.**
For example **if(parameter)** takes one parameter and decides whether to execute the next statement or block or not
- **Directory and File Names: Directory separators are automatically adapted to those used by the underlying operating system** Example: **C:/Test** changes to **C:\Test** automatically. **LINUX: \home\name\test** changes to **/home/name/test** accordingly.