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

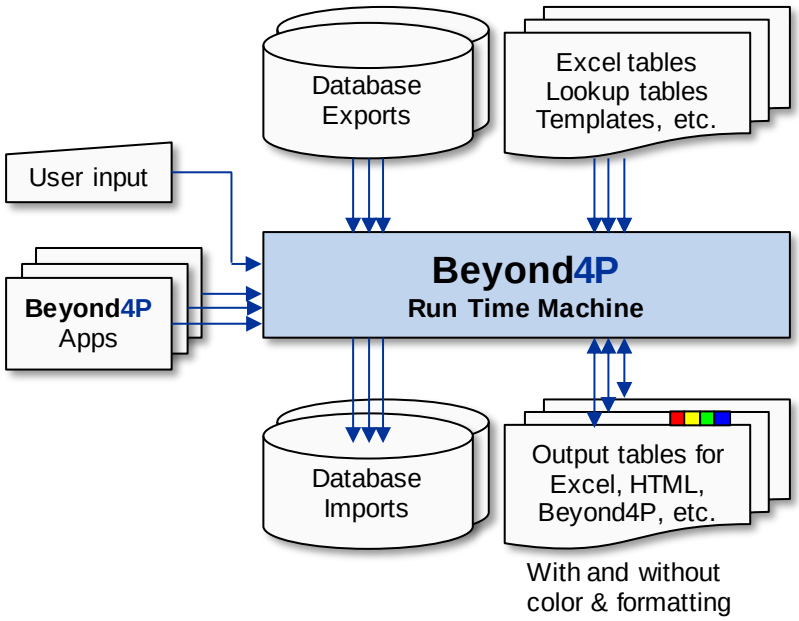
Cover photo: Office building with colorful window shades located in Schlieren near Zürich, Switzerland.

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

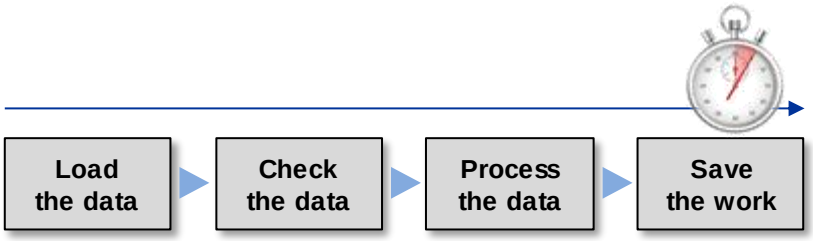
Release Notes	**7.04 – Various Improvements and Clean-ups**
Bugfixes	Loading JSON files: Crashed when encountering invalid literals. Double quotation marks inside strings to be used as part of strings were misinterpreted as ending strings. Both issues have been resolved. JSON: Previous version forgot about reading / writing empty arrays as they are different from *null*. 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. 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. Bugfixed and overhauled '**literal**' and '**table load**(… **HTML**) functions
New general features	If activated, **scientific notation** (exponents) is supported in accessing numbers in tables and **input** (…) function. **Automatic array creation**. E.g. if variable a[ ] is not yet existing or just a simple variable, then a[3] = 1 would create an array with 4 elements, assign 1 to the last one and initialize the remaining ones as void. a[5] would extend that array, i.e. initialize a[4] as void value and a[5] with the next value provided. **Github markdowns**: Files ending with ".md" will recognize code blocks (delimited with ``` symbols) and execute only those.
New functions	**table append on same field** (…) appends text contents to the last field in the last row. **text color** (…) and **background color** (…) changes colors on console **cursor** (…) sets cursor position on console **tokenize** (…) breaks a string down into tokens according to specified rules **table lookup smart once** (…) and **table lookup smart once ignore case** (…) combines advantages of both **table lookup smart** and **table lookup once** function families.
Enhanced functionality	**table save (…)** supports new format: TEXT. This assures original contents in tables saved as text without tampering with quotation marks. **replace** … (…) and **substitute** … (…) now allow replacing multiple old patterns by new patterns **input** (…): Numeric input now supports scientific notation
Performance enhancements	**Memory allocation and deallocation performance** has been strategized which results to higher performance when processing very large data.
Obsolescence	

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, .xlsm(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.