# WKAS/PL for Beginners

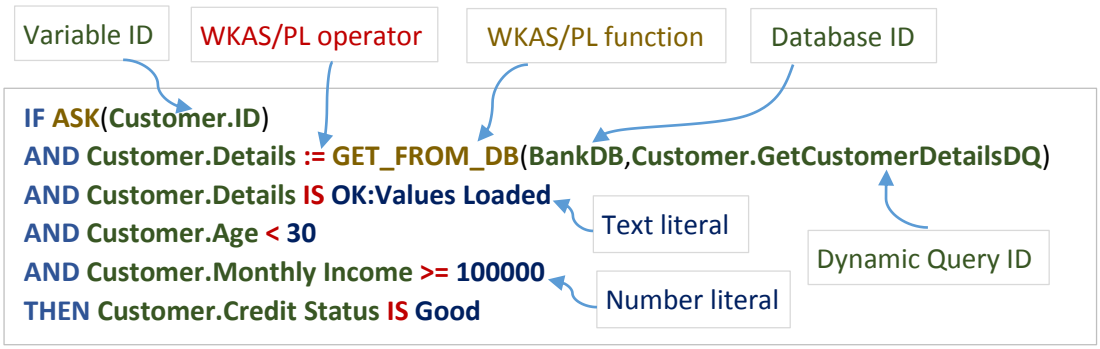
WKAS/PL is a 4GL rule-based language to develop web-based software, mobile web-apps and chat-bots backed by expert system (classical AI) technology. Many other technologies supported by WKA Studio (WKAS) such as genetic algorithms, case-based reasoning (facilitates lazy machine learning) can be programmatically invoked using WKAS/PL. WKAS apps use WKAS/PL in rules and user defined functions to implement procedural logic. The current version supports around 240+ functions and following data types. WKAS/PL provides lots of functions to do computations and write complex business logic involving various data types e.g. supporting matrix calculations and operations.

| | |
|---|---|
| Text | Holds text data. Spaces can be included when text is assigned, however, when language keywords, commas, brackets etc. are used, they must be entered in double quotes.<br><br>Student.Name:=S K Patil<br>Student.Grad Institute:= "Indian Institute of Technology, Bombay"<br><br>It can also hold XML data. If double quote appearing inside text, it has to be preceded by back-slash e.g. Customer.Statement:="He said \"this cannot be done\"" |
| Number | Hold the (32 bit) signed integer data.<br><br>Customer.Age:=50<br><br>Number without decimal point is treated as number. Number without decimal point is treated as number. Current version supports only 32bit, any number more than size of 32bit converted into BigNumber. |
| BigNumber | Hold the (64 bit) signed integer data.<br><br>Customer.Mobile:=808080808 |
| Real | Hold the float (32bit) data |

| | |
|---|---|
| | Student.Marks:=89.24; |
| Boolean | Boolean data types hold true or false.<br><br>Customer.HasJob:=true; |
| Compound | Holds list (array) of text, numbers, real, compound, trend, boolean, matrix etc.<br><br>Customer.Assets:= [Car, Two Wheeler, Colour TV, Washing Machine]. In case values themselves contain comma, then they can be entered in double quotes or pipe \| can be used as separator.<br><br>Any literal starting with '[' and ending with ']' would be treated as compound list. Compound variable can hold different types inside.<br><br>e.g. Customer.Record:=[P K Sing,40,"20,IIT Campus,Powai",[Pune,Mumbai]]<br>e.g. Customer.Choices:=[[Mumbai,1],[Chennai,2],[Delhi,4]]<br><br>Some of the compound variable can contain values with weights<br>Student.PLKnown:=[C++=100,Java=60,Python=50] |
| Date | Holds time, date or time stamp depending upon initial value<br><br>Student.DOB:=_0102200<br><br>Any literal starting with _ is treated as date type. Examples: _1030 would be treated as time 10:30am, _01012013 would be treated as DDMMYYYY (8 characters) format. Default formats can be changed/set using project config parameters: DefaultTimeFormat (must include colon separator HH:MN:SS), DefaultDateFormat (DD/MM/YYYY) and and DefaultDateFormatLong (DD/MM/YYYY HH:MN:SS) . If length of date string is more than length of formatting string set in DefaultDateFormat then long format is used to convert text into date |
| Matrix | Holds matrix data<br><br>Info.MatVar:={10 2 3,5 6 7} creates 2 x 3 matrix and assigns to the variable. Space (column separator) or comma (row separator) as a separator. Any literal starting with '{' and ending '}' is treated as matrix data type. However, CONVERT/READ_MATRIX function can be used to convert text into Matrix. |
| Trend | Holds the trend data. Array of numeric values (real). |

| | |
|---|---|
| | Info.Sales:=\|100 120 125 129 140\|, space as a separator. Any literal starting with \| and ending '\|' is treated as trend data type. Internally trend data type is stored as matrix with one column. |
| Document | This holds any type of document including images. Depending upon type of document like XML, etc. different treatment is given. |
| URL | This is same as Text data type except the value would be displayed as URL link when shown in the report. |
| Variant | Holds any data type |

## Sample Rule:




## Sample UDF (user defined function)

Name: Libraries.GetRDInterest

Parameters: [BankParams.MonthlyRDAmount, BankParams.IntRate, BankParams.PeriodInMonths]

Function Code:

```
BankParams.InterestAmount:=0
IF(BankParams.PeriodInMonths >=1 AND BankParams.MonthlyRDAmount>=1,Go)
    BankParams.InterestAmount := BankParams.MonthlyRDAmount * (1+BankParams.IntRate /400)^(1/3.0)
    BankParams.Count:=2
    WHILE(BankParams.Count<=BankParams.PeriodInMonths,RD)
        BankParams.InterestAmount := (BankParams.MonthlyRDAmount + BankParams.InterestAmount ) *
        (1 + BankParams.IntRate / 400)^(1/3.0)
        BankParams.Count:=BankParams.Count+1
    END_WHILE RD
END_IF Go
BankParams.InterestAmount := BankParams.InterestAmount-
BankParams.MonthlyRDAmount*BankParams.PeriodInMonths;
RESET_VAR_VALUE("BankParams.Count");
RETURN BankParams.InterestAmount;
```

AND is used as condition/statement separator and each condition must return Boolean data type. Space is not treated as separator in WKAS/PL. Following characters have specific meaning in WKAS/PL: comma (,), pipe (|), opening and closing brackets ([,],(,),{,}).

New line or semicolon (;) is used as statement separator in UDF.

## More about WKAS/PL

1. WKAS/PL uses lot of functions that use various objects created and managed using various WKAS interfaces. Each object has unique ID in that type. Examples are variable IDs, App IDs, DB IDs, menu IDs. These objects can be created and configured through respective interfaces. For example, new variable can be added through *Global Variables* Interface. There are quite a number of parameters which can be set for every variable created such as minimum, maximum and default values, validation script, input options like whether value is asked to the user, fetched from database or computed using some server side or JavaScript function whenever required etc. In case values to be asked to the user, web-pages are created with in-built validations and options using just a click of button, these web-pages are automatically invoked whenever inputs to variables are required. This saves lot of explicit coding of asking explicit inputs (calling database code in case, the value is to be fetched from the database), validating them, etc.

   Except few IDs like DB IDs, App IDs etc. most of the IDs are prefixed with application or variable group names e.g. *Customer.Age* where *Customer* is variable group and *Age* is variable name within variable group *Customer*, however,

*Customer.Age* is called as variable name/ID in general. ID names can contain spaces and underscores e.g. *Customer.First Name* or *Customer.First_Name*.

2. Seamless use of variables across client and database functionality. While writing JavaScript at client side, variable names can be included in square brackets which are replaced by JavaScript function e.g. validation script:

   if([Customer.Age]>60)
   DisplayMsg('Customer name should be less than or equal to 60');

   [Customer.Age] will be replaced by GetValue('Customer.Age').

   JS library provide various functions to manipulate data, controls in forms and reports.

   Similarly, database queries can include variable names e.g.

   select $[Customer.Age] from customer where $[Customer.ID]= [Customer.ID].

   $[Customer.Age] will be replaced by mapped field in table: customer and database and [Customer.Age] will be populated with value of variable *Customer.Age* at run-time.

3. Database functions help to get rid of explicit database coding e.g. LOAD_FROM_DB function a> opens DB connection using configured connection string defined through *Database*

*Connection* interface, b> executes predefined parameterized query by populating parameters run-time or direct query given with parameter placeholders in argument, c> fetches the data from the database and d> automatically populates (does the transformation of data if configured) data into respective variable objects based on mapping configuration); similarly, function UPDATE_TO_DB updates variable values into database without explicitly opening DB connection, creating update queries and executing them.

4. All functions are in CAPs and use underscore e.g. GET_FROM_DB: fetches data from database.

5. Spaces are not treated as separators.

6. Text literal need not be enclosed inside square brackets unless it contains keyword, function name or special characters (such as |,{},[]).

7. If any ID is appearing in the statement, the parser first checks whether it is variable ID or not, in case other IDs such as menu ID has same name as variable name, it should be included in double quotes to avoid conflict e.g. GET_MENU("Customer.Education", Code, BSc) in case there exists variable named as *Customer.Education*.

8. Since WKAS/PL is part of rule-based expert system technology, when working in expert system mode, execution engine asks variable value whenever needed. The functions which does not require values to be red, variables need to be included in double

quotes e.g. RESET_VAR_VALUE("Customer.Age"), this statement won't ask value of variable: *Customer.Age* but will reset its value when executed.

9. By default, values of variables used in the execution will be shown when goal is arrived. However, output of only specific variables can be shown by setting parameters for variable using Global Variable Interface.

## Steps to manually create simple app using WKA Studio that adds two numbers.

1. Create simple App using *Create & Manage Apps* interface say *AddApp* by giving application group (say myApps) and variable group (say *MathInfo*).

2. Add three input variables, *Number1*, *Number2* and *Result* in variable group say *MathInfo* as number data type using *Global Variables* Interface. Add options like min, max and default values etc. if required.

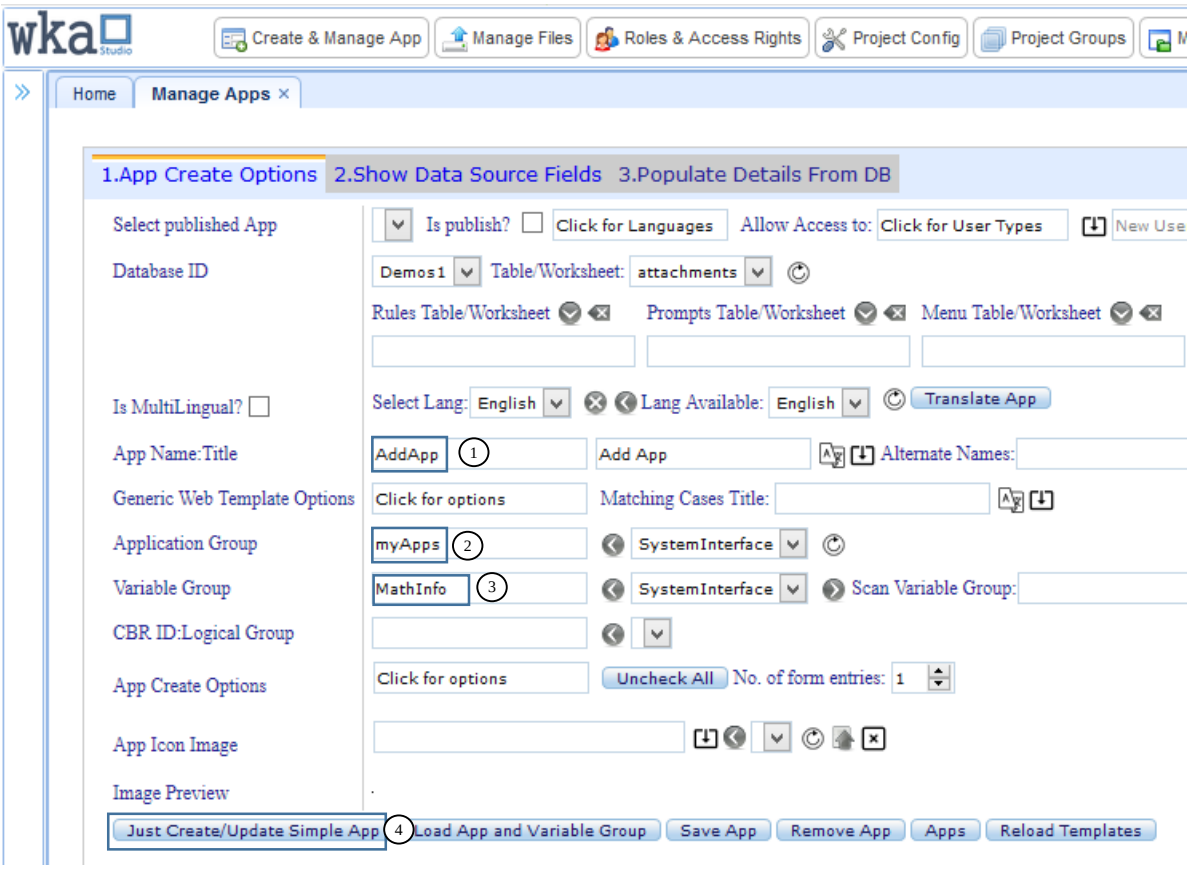
3. Just add statement: *MathInfo.Result := MathInfo.Number1 + MathInfo.Number2* in the default rule created by using *Rules/Expert System* Interface, update the rule and click on Create Web Pages by selecting app.

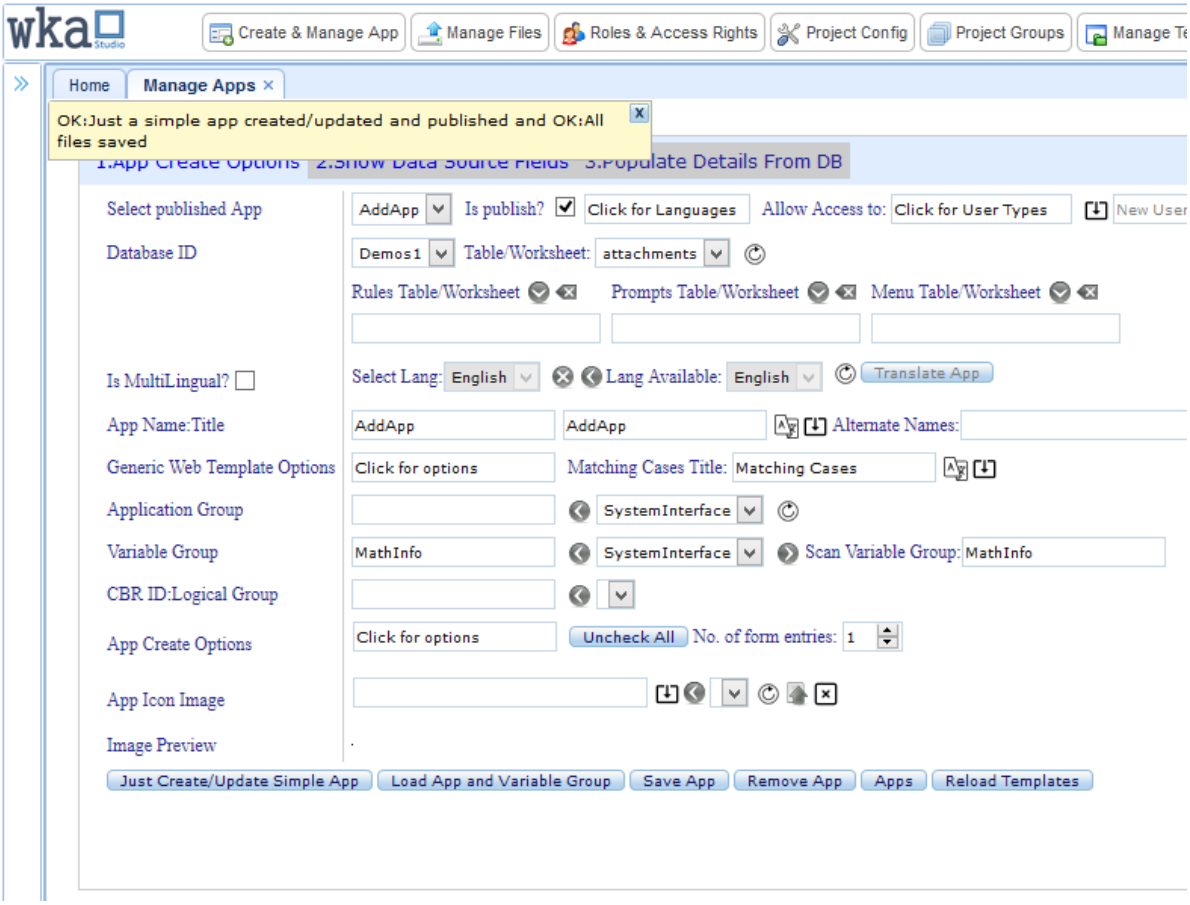
4. You are done! run the application through *Apps* interface

The above steps illustrate that developer just needs to write *one line of code* to create app that adds two numbers. Rest is managed through WKAS interfaces, explained in the following steps.

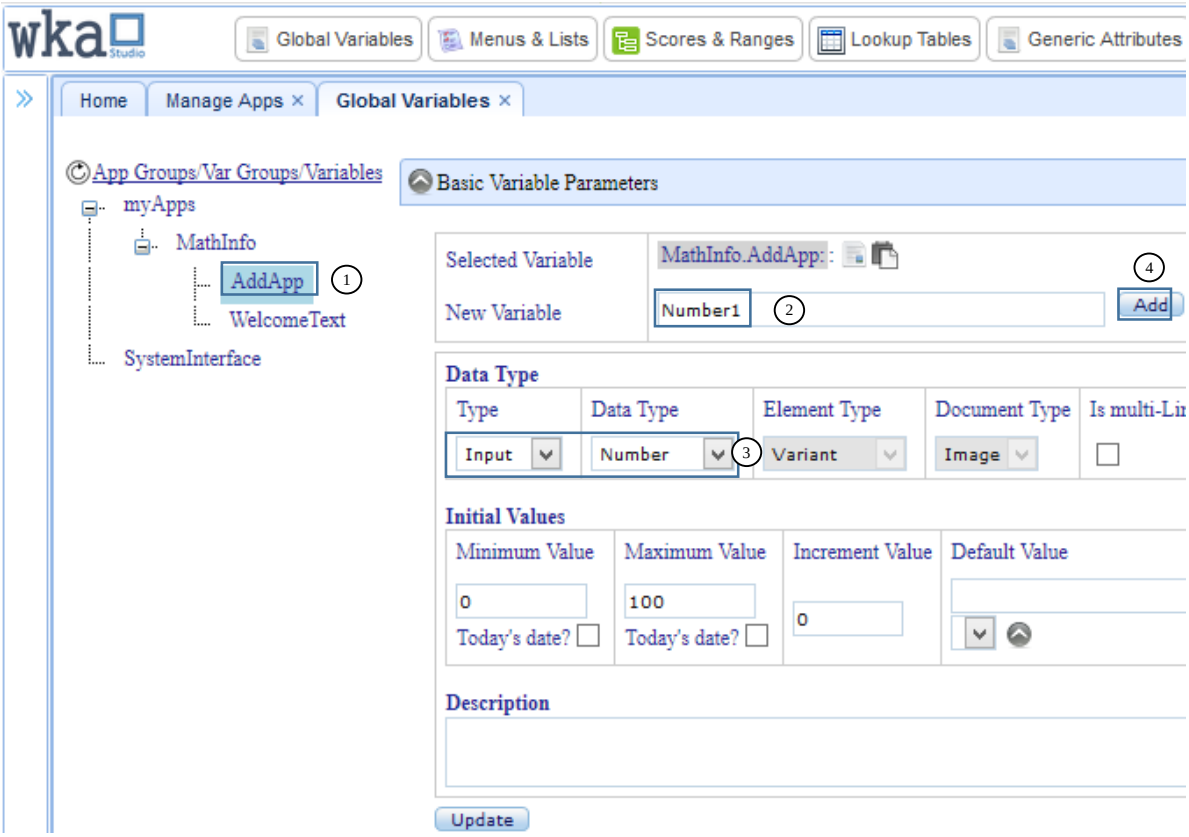
WKAS provides app creation interface to create apps where rules written in excel sheets can be imported. Using just few steps, all objects and entire application is created and does not to create objects explicitly (refer to tutorial section).

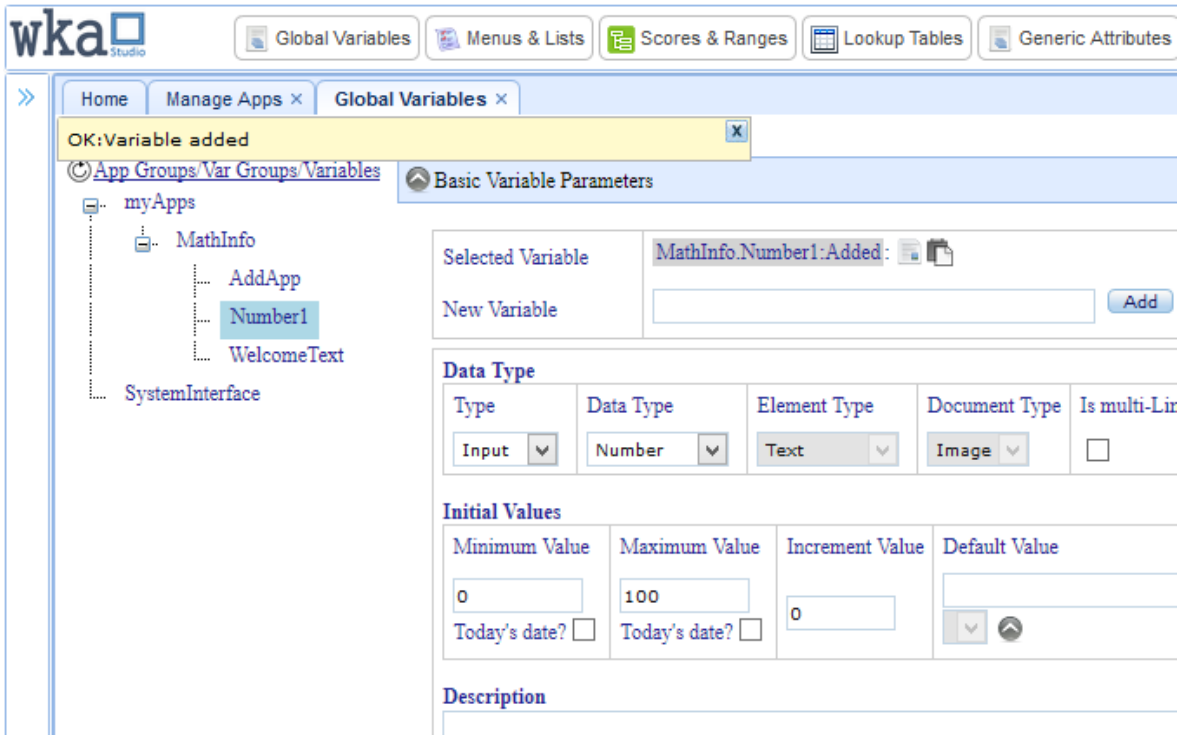
# Step 1: Creating app: sub-steps to be followed

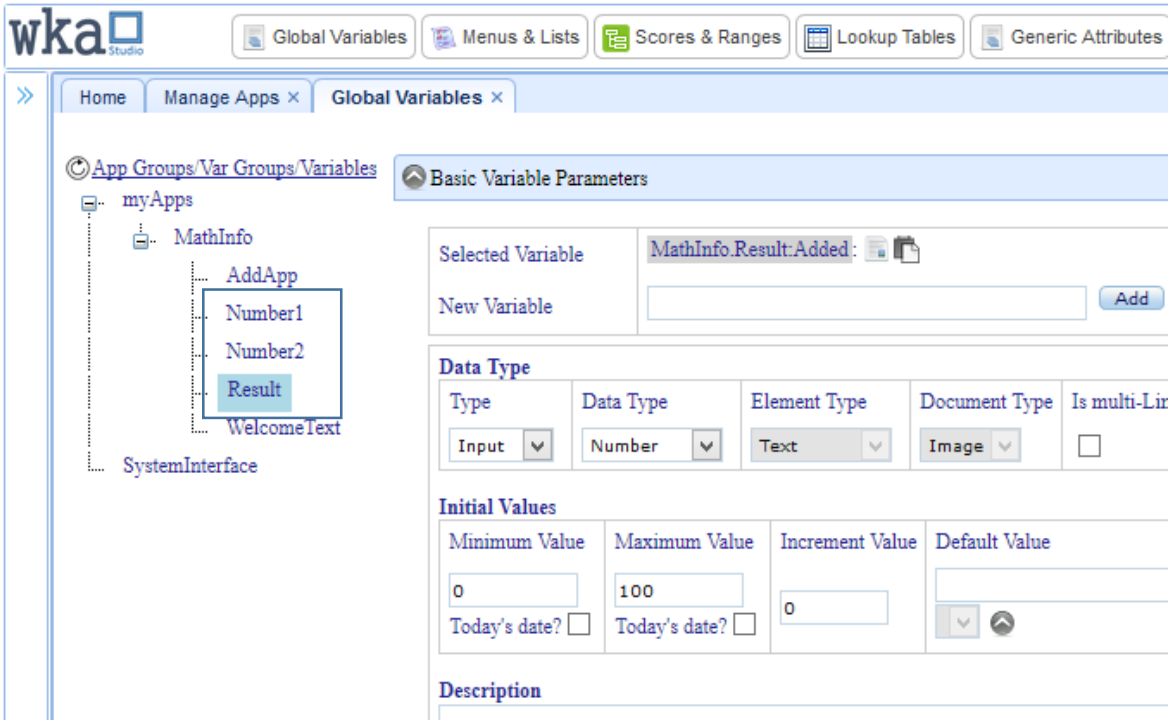


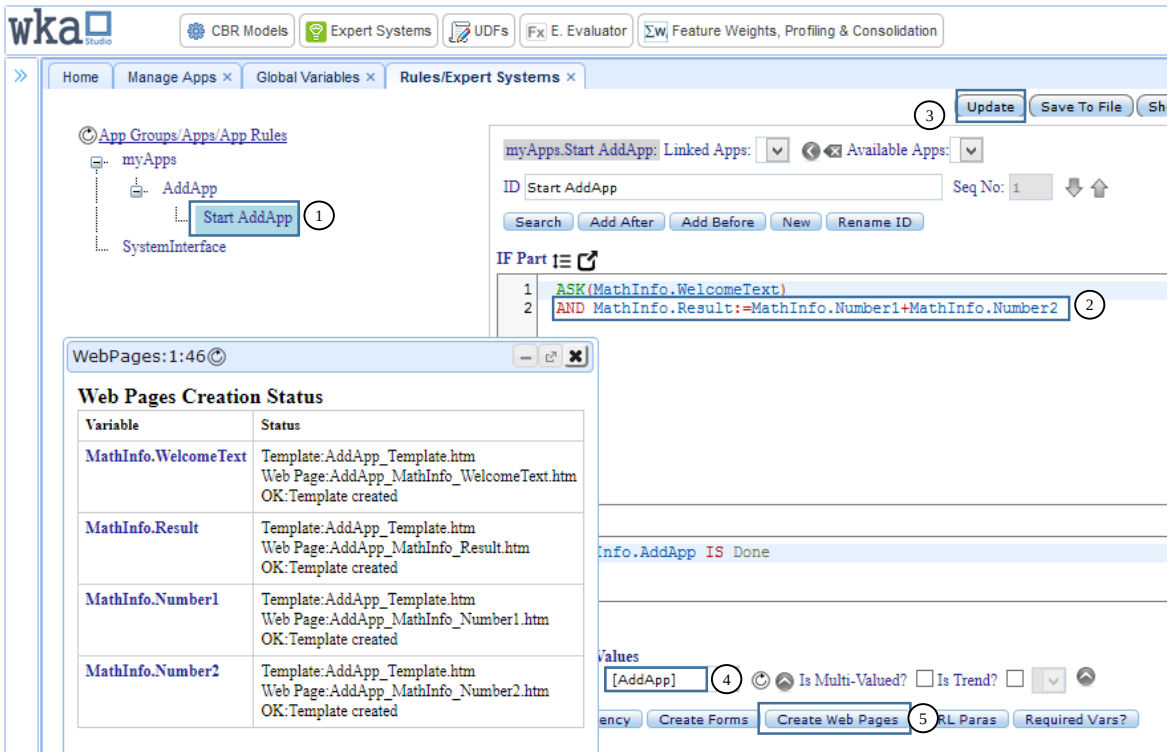
## Step 2: Add variables



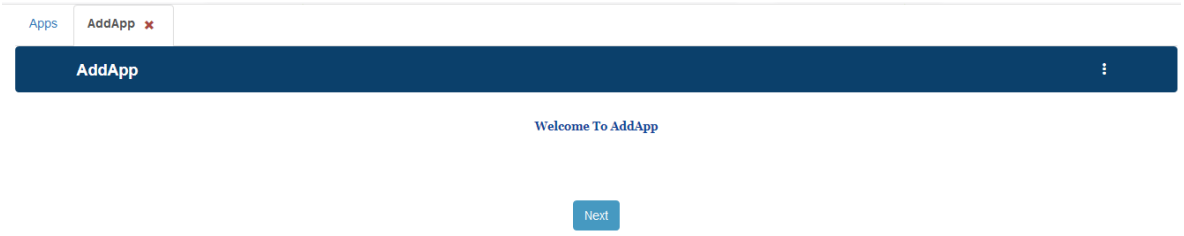



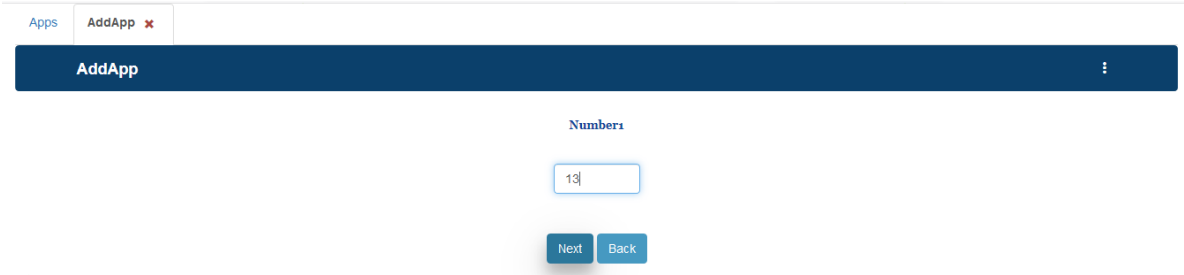


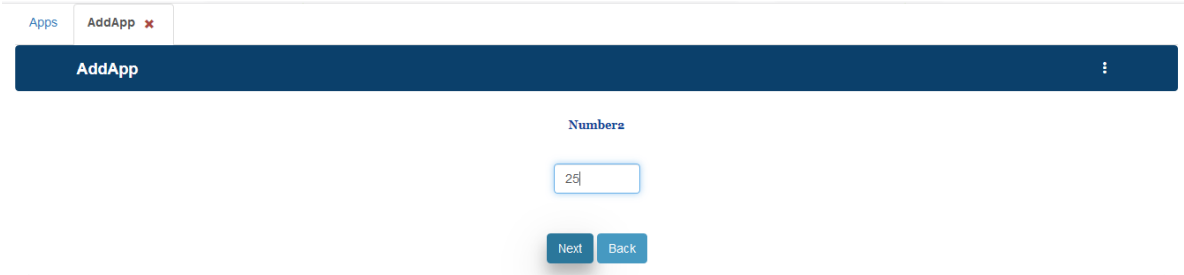

## Step 3: Modify default rule and adding statement

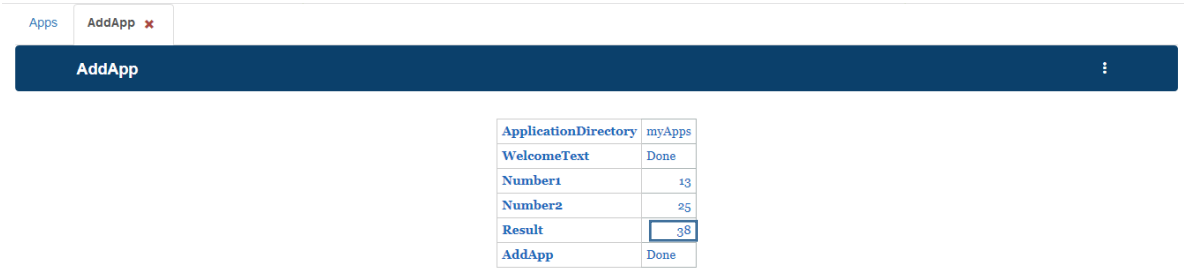


# Step 4: App Execution



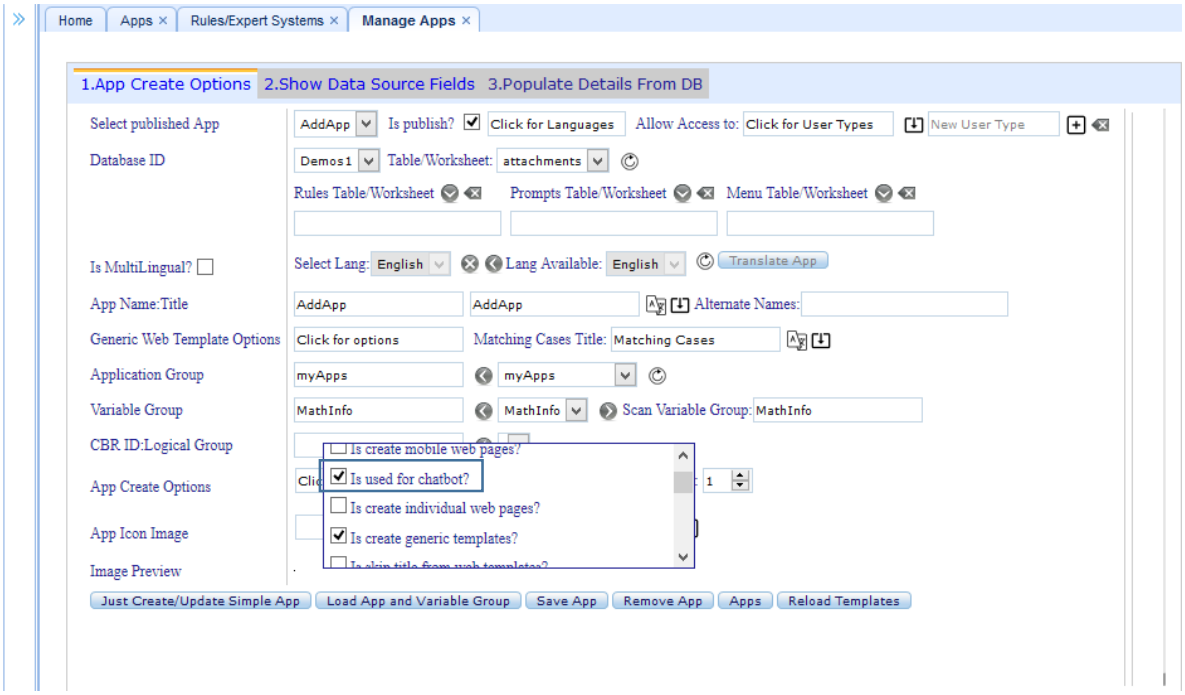



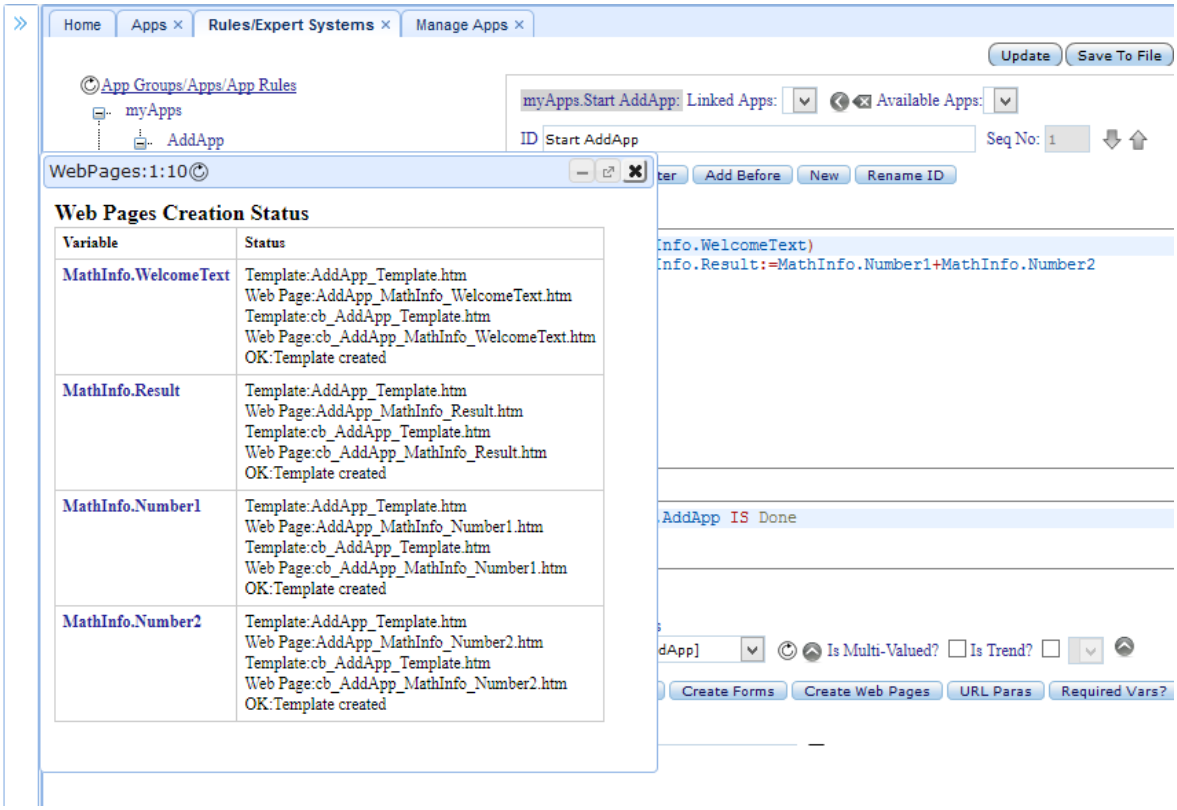



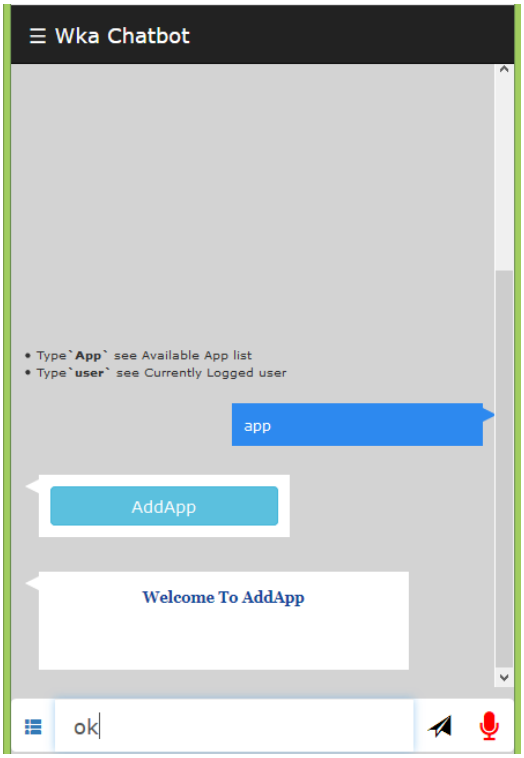
# Creating pages for Chat-bot interface

Select options ‘Is used for chatbot?’ and ‘Is overwrite/update to existing App?’

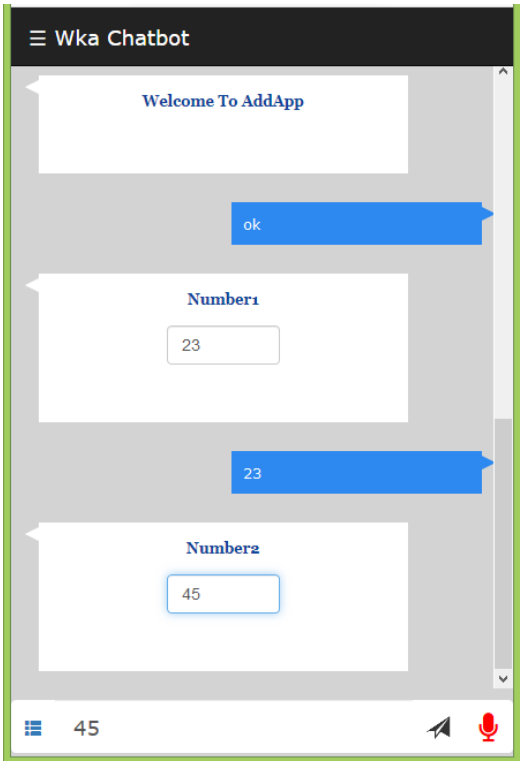


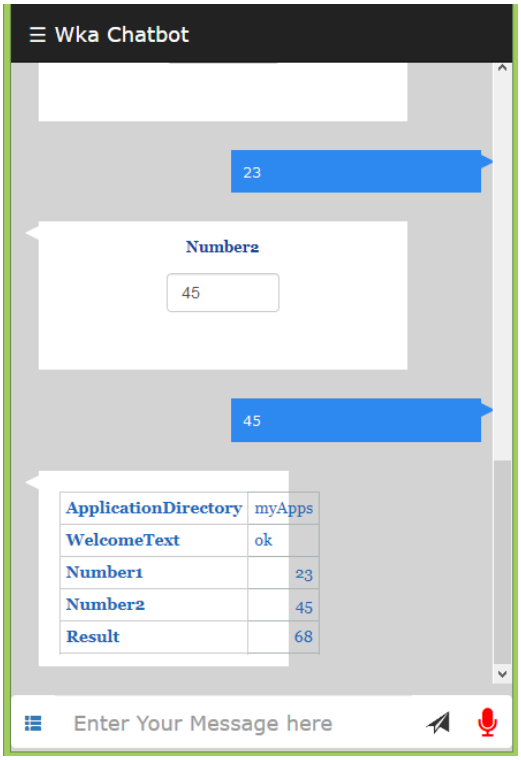

Create web-pages for chat-bot interface.










## WKAS/PL Reference