b4A White Paper

Coding Guidelines - Guidelines for modern development

best-blu consulting with energy GmbH

Your contact person:

E-mail:
Phone: +49 421 491 811 80
Date: May 27, 2025

Contents

Motivation

The need for automation has grown enormously in recent years. Not only has the number of automated processes increased, but the requirements have also risen and with them the complexity.

Many of the automation environments based on Automic Automation have been in companies for many years and are trying to meet the growing needs and requirements. With today's complexity, many of these environments are reaching their limits. There are many reasons for this. One point that is usually included is the growing importance and thus significance of automation systems and the processes running on them. Errors in the processes or other instabilities in these environments now often have fatal consequences for the most diverse areas of a company.

Major changes are needed to build an environment that meets these new requirements from old, established automation infrastructures and the associated development and test procedures. These must be applied in various areas in order to create a modern, stable and therefore high-quality automation platform.

This white paper deals with one of these topics and presents an exemplary solution. It deals with guidelines that define a basis for structured development and thus contribute to a stable foundation for modern automation.

Coding Guidelines

Many environments have been in the infrastructure for many years and are trying to adapt to the constant changes and growing requirements. The further development of standards in automation has often fallen by the wayside, making it increasingly difficult to keep up.

In order to bring high-quality and increasingly complex automation processes into production as quickly as possible, new concepts are needed in development, testing and for a deployment process. These must address the challenges and problems that exist today in many automated environments. An important pillar for this are the so-called *coding guidelines*. These are guidelines for development. In software projects, such rules are standard as soon as several people are involved. The aim of these rules is, for example, to speed up the understanding of solutions that have been implemented by other developers. There are many different options for implementing requirements. In many environments, you will therefore find a wide variety of solutions for one and the same problem. For developers, this means that they have to familiarize themselves with solutions developed by others more often than necessary. However, such a "free" form of development not only leads to increased effort in development, but can also make it more difficult to solve critical production problems and increase the resources required for maintenance.

Coding guidelines can also cover other topics that are important in the context of automation with the Automation Engine, even if they do not directly affect the implementation of the automation processes. One common topic is the definition of an object naming scheme. The object names are important for the authorization system and it is therefore important that there is a definition of how the objects are to be named. Parts of the object names are often used to define parameters for execution.

Another topic is the documentation of processes and individual tasks. On the one hand, this involves explanations of technical implementations and, on the other, technical descriptions such as instructions for operating.

All or only some of these topics, or even more, can be defined in the coding guidelines for the development of automation processes. The following is an example of rules that are used in the development of the b4A packages from best-blu consulting with energy GmbH.

Example

The following coding guidelines contain some rules that are only applicable if the b4A Package Management concept is used. These include, for example, the rules for the folder structure. In general, these rules are only to be understood as an example that can be used as inspiration for your own rules.

Folder

All packages that are published must be located in the following base folders

- `PACKAGES/BEST-BLU`
- `PACKAGES/BEST-BLU/API`
- `PACKAGES/BEST4AUTOMIC`
- `PACKAGES/BEST4AUTOMIC/BASICS`

Subfolder

- Necessary
  - `SERVICES` and/or `ACTIONS`: for the services and/or actions to be used directly
  - `CONFIG`: Configuration objects
  - `SOURCE`: all other objects. Subfolders can be created to divide the objects into logical groups (not by object type)
  - `DOCUMENTATION`: Release notes, package documentation and the like
- Optional
  - `RESOURCES`: For example, for storage objects

Naming schemes

Package

- `BBC.<function>` for general packages (e.g. BBC.MAIL, BBC.SAP, BBC.FILEOPS, ...)
- `B4A.<function>` for b4A specific packages (e.g. B4A.BASE, B4A.PM, B4A.TA, ...)
- `B4AB.<function>` for packages with basic functionalities (e.g. B4AB.TRANSPORT)
- `API.<product>` for packages that communicate with external interfaces (e.g. API.CONFLUENCE, API.JIRA, API.OWNCLOUD, ...)

Objects

- `<package>.<object type>[@<subtype>].<description>`
  - <subtype> should only be specified if there are any for the object type and this is not the default subtype. For example, the following types do not have this extension

        - Standard workflow (JOBP)
        - Static variable (VARA)
    - In the <description>, the '-' (minus) should be used as a separator between the words (minus) should be used

Package content

The following objects must exist

- `CONFIG/<Package>.VARA.SETTINGS` - Settings

The following fields must exist in the metadata variable `<PACKAGE>.VARA.METADATA`

- Name - Package name
- Version - Version number (three digits)
- Description - short description
- Dependencies - Dependency definition
- Supported - yes/no

The text documentation of <PACKAGE>.VARA.METADATA must contain the detailed description of the package.

Objects

Language

The language is English for:

- Object names
- Variables
- Documentation
- Comments

Configurations

The following attributes/values can be read from configuration variables ({....})

- Agents
- Logins
- Paths (file system)
- E-mail address
- Further environment-specific settings

Object title

Each object must have a title that describes the function

Object variables

(Inherited) object variables must begin with a prefix. This is composed as follows

- The first component of the package name
- 1 letter from the package name after the first dot
- An underscore (_)

Examples

- `BBC.MAIL → BBCM_`
- `BBC.INCIDENTS → BBCI_`
- `B4A.BASE B4AB_ →`

Input or output variables should have a special ending in their name. Such variables include all variables that can leave the context of the generating object or are intended to do so. Input variables are variables that come from a PromptSet. They should have the ending `_I#`. Variables that are used as a return value/output should use the ending `_O#`. Examples of this are

- Variables that are published using `:publish`
- the variable that contains the result of an EXEC variable
- Variables that are extracted from the response of a REST job
- Variables that represent the result of an include object

The ending `_L#` must be used for the run variable in FOREACH workflows.

Scripts (SCRI)

- Output and input variables should be capitalized
- local variables do not require an identifier in their name and can be written in lower case
- Object variables must follow the aforementioned identifiers Object variables

Includes (JOBI)

For replacement parameters, the pattern must start and end with a dollar($).

```
inc BBC.MAIL.JOBI.DOCU_TO_CALLTEXT $DOCU_OBJECT$ = "DOCU.NEW.1"
```

Documentation

The documentation of the objects takes place in the text documentation and is written in Markdown format (Basic Syntax | Markdown Guide). Depending on the object type, other sections are mandatory.

The following objects must have documentation:

- Workflows (JOBP) with PromptSet: The input parameters and the function of the workflow must be described
- Includes (JOBI): In addition to the function, the replacement parameter may have to be described and at least one example of the call must be specified
- Metadata variable: In this special variable, the scope of the b4A package must be described and the requirements defined. This includes necessary settings in the Automation Engine and also programs on the operating system
- Backend, Exec and SQL variables: This describes which data these variables return by column and which input variables are required

Launchable objects with PromptSets (e.g. actions)

```
### Description

A short description of the functionality

### Parameters
```

```
Param1
: description of param1

Param2
: description of param2

### Output

&BBCX_VARIABLE_O#
: description of the variable
```

Include (JOBI)

```
### Description

A short description of the functionality

### Parameters

$PARAM1$
: description of the parameter

### Input

&BBCX_VARIABLE_I#
: description of the variable

### Output

&BBCX_VARIABLE_O#
object name ended for usage in URLs

### Example

    include BBC.PACKAGE.JOBI.MY_INCLUDE $PARAM1$ = "<value>"
```

Metadata variable

```
This package provides utilities for starting b4A modules and creating
services using several b4A modules. All actions can be used on
Unix/Linux and Windows systems.

### Requirements

* Unix/Linux

  * a bash 2 compatible shell
  * Unix tools: cat, rm, mktemp
```

```
* Windows

 * Powershell 7.0
```

BACKEND, EXEC and SQL variables

```
### Description

Retrieves the sub-type of a given object

### Input

&BBCX_VARIABLE_I#
: description of the input parameter

### Output

Column | Name        | Description
------ | ----------- | --------------------
1      |  Objectname | Name of the object
2       | Type       | Type of the object
3       | Subtype    | Subtype of the object
```

Conclusion

Such rules help development teams to carry out their work more efficiently while maintaining consistent quality. In operating, it is easier to carry out error analyses and thus solve production problems more quickly.

Especially through the rain for the documentation, implementations can be understood more quickly by others and adaptations or extensions can be carried out more quickly. With the *b4A Documentation Builder*, the documentation can also be transferred to other systems such as simple websites or wiki systems and made available to other departments.

With the *b4A Compliance Check*, it is also possible to check many of these rules fully automatically. This avoids human error and prevents automation processes that do not adhere to the compliance rules from ever entering production.