b4A White Paper

Package Management – Der Weg zu einem Deployment-Verfahren

best-blu consulting with energy GmbH

Ihr Ansprechpartner:
Andreas Büsching

E-Mail: a.buesching@best-blu.de
Telefon: +49 421 491 811 80
Datum: 27. Mai 2025

Inhalt

Motivation

Der Bedarf an Automation ist in den letzten Jahren enorm gewachsen. Dabei hat sich nicht nur die Menge der automatisierten Prozesse erhöht, sondern auch die Anforderungen sind gestiegen und damit ebenso die Komplexität.

Viele der Automationsumgebungen auf Basis von Automic Automation sind seit vielen Jahren in den Unternehmen und versuchen sich den wachsenden Bedürfnissen und Anforderungen zu stellen. Mit der heutigen Komplexität kommen viele der Umgebung an ihre Grenzen. Gründe hierfür sind vielseitig. Ein Punkt, der in der Regel dazu gehört, ist die wachsende Wichtigkeit und damit Bedeutung der Automationssysteme und der darauf laufenden Prozesse. Fehler in den Prozessen oder andere Instabilitäten dieser Umgebungen haben mittlerweile oft fatale Auswirkungen auf die unterschiedlichsten Bereiche eines Unternehmens.

Um aus alten gewachsenen Automationsinfrastrukturen und den zugehörigen Entwicklungs- und Testverfahren eine Umgebung zu bauen, die diesen neuen Anforderungen gerecht wird, bedarf es größeren Änderungen. Diese müssen in verschiedenen Gebieten angewendet werden, um eine moderne, stabile und damit qualitativ hochwertige Automationsplattform zu schaffen.

Dieses White Paper beschäftigt sich mit einem dieser Themengebiete und zeigt hierfür eine beispielhafte Lösung auf. Dabei geht es um Richtlinien, die eine Grundlage für eine strukturierte Entwicklung festlegen und damit zu einer stabilen Basis für eine moderne Automation beitragen.

Coding Guidelines

Viele Umgebungen befinden sich schon viele Jahre in der Infrastruktur und versuchen sich den ständigen Änderungen und wachsenden Anforderungen anzupassen. Hierbei ist die Weiterentwicklung von Standards in der Automation häufig auf der Strecke geblieben, wodurch ein Mithalten immer schwieriger wird.

Um qualitativ gute und immer komplexere Automationsprozesse möglichst schnell in die Produktion zu bringen, bedarf es neuer Konzepte in der Entwicklung, beim Testen und für ein Deployment-Verfahren. Diese müssen sich den Herausforderungen und Problemen widmen, die heutzutage in vielen Automic Automation Umgebungen existieren. Ein wichtiger Pfeiler hierfür sind die sogenannten *Coding Guidelines*. Darunter versteht man Richtlinien für die Entwicklung. In Software-Projekten sind solche Regeln Standard so bald mehrere Personen beteiligt sind. Ziele dieser Regeln sind beispielsweise das schnellere Verstehen von Lösungen, die durch andere Entwickler implementiert worden sind. Bei der Umsetzung von Anforderungen, gibt es viele verschiedene Möglichkeiten. In vielen Umgebungen findet man daher auch die unterschiedlichsten Lösungsvarianten für ein und dasselbe Problem. Für Entwickler heißt dies, dass sie sich öfter als notwendig in Lösungen einarbeiten müssen, die von anderen entwickelt worden sind. Solch eine „freie“ Form der Entwicklung führt allerdings nicht nur in der Entwicklung zu erhöhten Aufwänden, sondern kann auch das Lösen von kritischen Produktionsproblemen erschweren und die notwendigen Ressourcen für die Maintenance erhöhen.

Coding Guidelines können zusätzlich weitere Themen abdecken, die im Kontext der Automation mit der Automation Engine wichtig sind, auch wenn sie nicht direkt die Implementierung der Automationsprozesse betreffen. Ein häufiges Thema ist die Definition eines Objektnamenschema. Die

Objektnamen sind wichtig für das Berechtigungssystem und daher ist es wichtig, dass hier eine Definition existiert, wie die Objekte zu benennen sind. Häufig werden Teile der Objektnamen genutzt, um Parameter für die Ausführung zu definieren.

Ein weiteres Thema ist die Dokumentation der Prozesse sowie der einzelnen Aufgaben. Hier geht es einerseits um Erklärung zu technischen Umsetzungen als auch um die fachliche Beschreibung wie Instruktionen für das Operating.

Alle oder nur einige dieser Themen oder sogar noch weitere können für die Entwicklung von Automationsprozessen in den Coding Guidelines definiert werden. Im Folgenden ist ein Beispiel von Regeln zu finden, die bei der Entwicklung der b4A Packages von best-blu consulting with energy GmbH angewendet werden.

Beispiel

Die folgenden Coding Guidelines enthalten einige Regeln, die nur anwendbar sind, wenn das Konzept vom b4A Package Management angewendet wird. Dazu gehören beispielsweise die Regeln für die Ordnerstruktur. Generell sind diese Regeln nur als ein Beispiel zu verstehen, das als Inspiration für eigene Regeln genutzt werden kann.

Ordner

Alle Packages, die veröffentlicht werden, müssen in den folgenden Basisordnern liegen

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

Unterordner

- Notwendig
  - `SERVICES` und/oder `ACTIONS`: für die direkt zu nutzenden Services und/oder Actions
  - `CONFIG`: Konfigurationsobjekte
  - `SOURCE`: alle sonstigen Objekte. Es können Unterordner angelegt werden, die die Objekte in logische Gruppen unterteilen (nicht nach Objekttypen)
  - `DOCUMENTATION`: Release Notes, Package Dokumentation und ähnliches
- Optional
  - `RESOURCES`: Beispielsweise für Storage-Objekte

Namenschemata

Package

- `BBC.<Funktion>` für allgemeine Packages (z.B. BBC.MAIL, BBC.SAP, BBC.FILEOPS, ...)
- `B4A.<Funktion>` für b4A spezifische Packages (z.B. B4A.BASE, B4A.PM, B4A.TA, ...)
- `B4AB.<Funktion>` für Packages mit Basisfunktionalitäten (z.B. B4AB.TRANSPORT)
- `API.<Produkt>` für Packages die mit externen Schnittstellen kommunizieren (z.B. API.CONFLUENCE, API.JIRA, API.OWNCLOUD, …)

Objekte

- `<Package>.<Objekttyp>[@<Subtyp>].<Beschreibung>`
    - <Subtyp> ist nur anzugeben, wenn es welche gibt für den Objekttyp und dies nicht der Standard-Subtyp ist. Beispielsweise haben folgende Typen diese Erweiterung nicht
        - Standard Workflow (JOBP)
        - Statische Variable (VARA)
    - In der <Beschreibung> soll als Trennzeichen zwischen den Wörter das '-' (Minus) genutzt werden

Package-Inhalt

Folgende Objekte müssen existieren

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

Folgende Felder in der Metadaten-Variable `<PACKAGE>.VARA.METADATA` müssen existieren

- Name – Package-Name
- Version – Versionsnummer (dreistellig)
- Description - kurze Beschreibung
- Dependencies – Abhängigkeitsdefinition
- Supported - yes/no

Die Textdokumentation von <PACKAGE>.VARA.METADATA muss die ausführliche Beschreibung des Package beinhalten.

Objekte

Sprache

Die Sprache ist Englisch für:

- Objektnamen
- Variablen
- Dokumentation
- Kommentare

Konfigurationen

Folgende Attribute/Werte sind aus Konfigurationsvariablen zu lesen ({....})

- Agenten
- Logins
- Pfade (Dateisystem)
- E-Mail-Adresse
- weitere umgebungsspezifische Einstellungen

Objekttitel

Jedes Objekt muss einen Titel haben, der die Funktion beschreibt

Objekt-Variablen

(Vererbte) Objekt-Variablen müssen mit einem Präfix beginnen. Diese setzt sich wie folgt zusammen

- Die erste Komponente des Package-Namen
- 1 Buchstabe aus dem Package-Namen nach dem ersten Punkt
- Ein Unterstrich (_)

Beispiele

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

Eingabe- oder Ausgabevariablen sollen eine spezielle Endung im Namen haben. Zu solchen Variablen gehören alle, die den Kontext des erzeugenden Objektes verlassen können bzw. dafür gedacht sind. Eingabevariablen sind Variablen, die aus einem PromptSet kommen. Die sollten die Endung `_I#` bekommen. Variablen, die als Rückgabewert/Ausgabe genutzt werden, sollen die Endung `_O#` nutzen. Beispiele dafür sind

- Variablen, die mittels `:publish` veröffentlicht werden
- die Variable, die das Ergebnis einer EXEC-Variable enthält
- Variablen, die aus der Antwort eines REST-Job extrahiert werden
- Variablen, die das Ergebnis eines Include-Objektes repräsentieren

Für die Laufvariable in FOREACH-Workflows ist die Endung `_L#` zu nutzen.

Scripte (SCRI)

- Ausgabe- sowie Eingabe-Variablen sollen großgeschrieben werden
- lokale Variablen benötigen keine Kennung im Namen und können klein geschrieben werden
- Objekt-Variablen müssen den zuvor genannten Kennungen folgen Objekt-Variablen

Includes (JOBI)

Bei Ersetzungsparametern muss das Muster mit einem Dollar($) anfangen und enden.

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

Dokumentation

Die Dokumentation der Objekte erfolgt in der Text-Dokumentation und wird im Markdown-Format (Basic Syntax | Markdown Guide) geschrieben. Je nach Objekttyp sollen andere Abschnitte verpflichtend enthalten sein.

Folgende Objekte müssen eine Dokumentation haben:

- Workflows (JOBP) mit PromptSet: Es müssen die Eingabeparameter sowie die Funktion des Workflows beschrieben werden
- Includes (JOBI): Zusätzlich zur Funktion muss ggf. der Ersetzungsparameter beschrieben werden und es ist mindestens ein Beispiel für den Aufruf anzugeben
- Metadaten-Variable: In dieser speziellen Variable muss der Umfang des b4A Package beschrieben und die Anforderungen definiert werden. Dazu gehören notwendige Einstellungen in der Automation Engine und auch Programme auf dem Betriebssystem
- Backend-, Exec und SQL-Variablen: Hier ist zu beschrieben welche Daten nach Spalten diese Variablen zurückliefern und welche Eingabevariablen benötigt werden

Startbare Objekte mit PromptSets (z.B. Actions)

```
### Description

A short description of the functionality

### Parameter

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

### Parameter

$PARAM1$
: 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>"
```

Metadaten-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- und SQL-Variablen

```
### 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
```

Fazit

Solche Regeln helfen Entwickler-Teams ihre Arbeit effizienter umzusetzen und dabei eine gleichbleibende Qualität zu wahren. Im Operating ist es einfacher Fehleranalysen durchzuführen und somit Produktionsprobleme schneller zu lösen.

Speziell durch die Regen für die Dokumentation können Implementierungen von anderen schneller verstanden werden und Anpassungen oder Erweiterungen sind schneller durchführbar. Mit dem *b4A Documentation Builder* kann die Dokumentation auch in andere Systeme wie beispielsweise einfache Webseiten oder Wiki-Systeme überführt werden so auch anderen Fachbereichen zur Verfügung gestellt werden.

Mit dem *b4A Compliance Check* ist es zusätzlich möglich viele dieser Regeln vollautomatisch zu prüfen. Dadurch können menschliche Fehler vermieden werden und verhindert werden, dass Automationsprozesse, die die Compliance Regeln nicht einhalten, jemals in die Produktion kommen.