Integrating IKAN ALM and Mainframes

Cost-effective and easy to implement Enterprise-wide ALM for both mainframe and non-mainframe environments
Table of contents

IKAN ALM: The Ideal ALM Solution for the Mainframe ................................................................. 4
IKAN ALM – User’s Point of View .................................................................................................... 4
  Step1: Log on and display the Desktop Overview .................................................................... 5
  Step2: Create/update a package and link it to a Project Stream ............................................. 6
  Step3: Select the required Action ............................................................................................ 9
  Step4: Create the Level Request .............................................................................................. 9
  Alternative Way: Commandline Interface (CLI) ........................................................................ 10
  Additional information provided by IKAN ALM ...................................................................... 11
  What happens behind the scene? ............................................................................................. 14
IKAN ALM – Administrator’s Point of View ................................................................................. 20
  Step1: Create the global Phases ............................................................................................. 20
    The Phase concept .............................................................................................................. 20
    The IKAN ALM Phase Structure ....................................................................................... 20
    The Common Files ............................................................................................................. 20
    The Resource Files ............................................................................................................. 21
    The Model files .................................................................................................................. 23
    The compileCobol_jcl.model ......................................................................................... 24
    An IKAN ALM phase and its usage: the z/OS compile phase ........................................... 25
    An IKAN ALM phase and its usage: the z/OS deployment phase ....................................... 27
  Step2: Create the IKAN ALM project(s) .................................................................................. 28
  Step3: Adapt the lifecycle (if necessary) .................................................................................. 29
  Step4: Define the environments and the necessary parameters ............................................. 30
  Step5: Add phases .................................................................................................................. 31
  Step6: Modify the phase parameters ....................................................................................... 32
Conclusion ..................................................................................................................................... 34
Related Document ..................................................................................................................... 34
Appendix I: IKAN ALM Terminology .......................................................................................... 35
Appendix II: CA-ENDEVOR Terminology .................................................................................... 36
Appendix III: Serena ChangeMan ZMF terminology ................................................................. 38
Appendix IV: Available z/OS IKAN ALM Phases ......................................................................... 39
Appendix V: Migration to IKAN ALM ....................................................................................... 41
Appendix VI: Sample of z/OS compilation JCL .......................................................................... 41
Summary

This technical document is intended for developers, technical people, mainframe or non-mainframe experts, and software architects.

IKAN ALM is a web-based Application Lifecycle Management tool. It combines continuous integration and lifecycle management, offering a single point of control and delivering support for build and deploy processes (manually generated or automated), approval processes, release management, and software lifecycles. IKAN ALM tightly integrates with leading third-party versioning solutions, build and deploy tools, and issue tracking software. It supports both mainframe and non-mainframe systems and, in case of mixed environments, it will handle the dependencies between both systems.

This document aims at explaining how, by using IKAN ALM, you can manage your application lifecycle, be it on mainframe or on distributed systems or on a combination of the two, and how you can easily deploy the developed applications on the mainframe.

We will describe in detail how IKAN ALM works and what the different tasks are for Users and Administrators.

We are confident that after having read this document, you will be convinced of the enormous advantages of putting in place our IKAN ALM solution.

If you would still have questions, do not hesitate to contact our support team at info@ikanalm.com.

Remark: Although IKAN ALM supports many types of mainframes (IBM, Fujitsu, Unisys, Bull,...), we will use IBM z/OS as an example throughout this document.
IKAN ALM: The Ideal ALM Solution for the Mainframe

In the following section, we will explain more in detail how IKAN ALM handles the lifecycle to compile and deploy your applications on the mainframe.

Today, traditional mainframe development is already often enhanced with Eclipse-based development to address the requirements of modern end-user applications. The main issue when combining mainframe and distributed development, is how to deploy the developed applications on the mainframe.

We will cover 2 points of view: the IKAN ALM User and the IKAN ALM Administrator.

IKAN ALM – User’s Point of View

Once the initial setup has been done and the projects have been set up by the IKAN ALM Administrator, a User can start using IKAN ALM.

Basically, a User can create a Compile/Build or Deploy action (a “Level Request” in IKAN ALM terminology) in 4 steps:

1. Log on to IKAN ALM and display the Desktop Overview
2. Create or update a package and link it to a Project Stream
3. Select the appropriate Action (Compile/Build or Deploy)
4. Create the Level Request

Once the Level Request is created, a series of information screens will be available to provide additional information on the requested action, allowing following up its status and the results.

A User creates/updates packages to compile and deploy (promote) one or more programs he developed.

The initial IKAN ALM setup for the mainframe and the project setup are done by the IKAN ALM Administrator.
Step 1: Log on and display the Desktop Overview

Next, the Desktop Overview will be displayed showing the list of Project Streams or Packages the User is working with. This Desktop can be completely customized.

The following basic information will be displayed:

- **Project type**: release-based or package-based
- **Project Stream (package) name**
- **Defined Levels**:
  A level is a logical step in the application lifecycle. The available levels are: Build (Compile), Test and Production. One or more of each of those levels can be used to define a lifecycle.
• **Next Scheduled Request:**
  If a Schedule was assigned to a Level (like in continuous integration) this field contains the execution date and time of the next request.

• **Latest Level Request:** shows the status of the latest request, the VCR tag and timestamp.

• **Latest successful Level Request:** shows the latest successful level request.

• **Action:** the available action icons for the Level. When clicking an action button, a level request will be started.

• **Message:** if it is not possible to define a request for a specific level, this message indicates the reason.

The z/OS project we use here as an example is a package-based project for which the following Levels have been defined: a Build Level (BUILDZOS) and some Deploy Levels (ZOSTEST, MILESTONE and PRODZOS). Those Levels are linked to Lifecycle(s), and the Project Stream(s) (i.e., the Head or a Branch) is/are also linked to a Lifecycle.

For mainframe use, a project must be package-based. A package allows the User to select one or several components of a Project Stream which should be built and deployed together, ignoring the other Project Stream components. Such a package has to stay coherent for building and deploying. A Package (of components) will always be linked to a Project Stream (in our example: ZOSDEMO H_1.0 Pack-001) and it will always follow that Project Stream's lifecycle. Also, a package has to progress with the Project Stream's lifecycle independently of possible other packages linked to the same Project Stream.

For distributed release-based projects, on the other hand, all components are built and deployed together.

### Step 2: Create/update a package and link it to a Project Stream

Before compiling/building, the User has to create a package that contains the sources he wants to compile/build and the copybooks.

1. **First of all,** the User has to specify the link to the correct Project Stream.
2. Next, he has to specify the name and description for the Package.

3. Once that is done, the User can select the required programs. IKAN ALM will display all the files available for the selected Project Stream. The User can select the files and indicate the required revision number. Many types of files can be selected, built and deployed in the same process (i.e., JCLs, Procs, Maps, Sysin, SQL, Sources with Assemble, COBOL, PL/1 languages, IDMS entities,..).

The following figure shows the package information before selection.
The next figure shows the package information after the selection of the required files.

Once the IKAN ALM User has defined the package, he can start building/compiling and, next, promoting or deploying his programs.
Step 3: Select the required Action

To execute a build/compile or deploy, the User can simply click the required action button in the Action column.

Step 4: Create the Level Request

To start a compile for the project in our example, the User would click the appropriate Level Request action button at the Build Level. Next, the Create Level Request screen will be displayed.
On this screen, the User can enter a description, view the parameters linked to the level request and, if configured that way, change some parameter values.

By clicking the Create button, the level request will be created and the process will start.

⚠️ In fact, that is all a User needs to do for compiling all the programs in his package: go to his Desktop, click the appropriate action button and create a Level Request to execute a Build/Compile or a Deploy.

**Alternative Way: Commandline Interface (CLI)**

Another possibility to create a Compile/Build or Deploy action is to use the Commandline Interface (CLI) and to use the external tool configuration from Eclipse to configure the Level Request from within Eclipse.
Additional information provided by IKAN ALM

Besides creating a Level request, IKAN ALM also provides a lot of additional information to a User:

- An overview of each Level Request.
- For each Level Request: detailed information through the interface.
- IKAN ALM reports. IKAN ALM comes with predefined reports, but the User can also define his own reports.

On the following pages we will show some sample IKAN ALM screens with more detailed information, such as the detailed Level Request information, the Build Log, the used Parameters, a sample report, a sample notification e-mail and the integration with an Issued Tracking System.

Finally, we will also show what happens behind the scene on the mainframe.
Integrating IKAN ALM and Mainframes

Phase Log

Phase z/OS program Compilation
Start Date/Time: 2013-10-16 22:00:00.0
End Date/Time: 2013-10-16 22:21:24.0
Duration: 00:01:18
Status: Success

Message:
Script Execution successful.
Execution results in: D:/ikan/ALM_environments/contbuild/target/382

Log

Status: SUCCESSFUL
Total Time: 1 minute 5 seconds

Events

<table>
<thead>
<tr>
<th>File Properties</th>
<th>Command</th>
</tr>
</thead>
</table>
| addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project. | addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project.
| addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project. | addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project.
| addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project. | addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project.
| addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project. | addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project.
| addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project. | addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project.
| addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project. | addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project.
| addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project. | addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project.
| addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project. | addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project.
| addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project. | addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project.
| addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project. | addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project.
| addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project. | addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project.
| addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project. | addPropertiesFromBuild | [move] Adding property COM.ibm.analyser_project_properties file found in this project.

View Build Log

Used Build Parameters

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>alm.build.environmentName</td>
<td>BUILDZOS</td>
</tr>
<tr>
<td>alm.build.filename</td>
<td>ZOSDEMO_M_1-0_Pack-001_b1_BUILDZOS_#1.zip</td>
</tr>
<tr>
<td>alm.build.machineName</td>
<td>herculexp</td>
</tr>
<tr>
<td>alm.build.number</td>
<td>3</td>
</tr>
<tr>
<td>alm.build.id</td>
<td>382</td>
</tr>
<tr>
<td>alm.build.startDateTime</td>
<td>16-10-2013 22:19</td>
</tr>
<tr>
<td>alm.build.actionType</td>
<td>Requested build</td>
</tr>
<tr>
<td>alm.build.requestType</td>
<td>Full Build</td>
</tr>
<tr>
<td>alm.build.requestId</td>
<td>BUILDZOS</td>
</tr>
<tr>
<td>alm.build.request.startDateTime</td>
<td>16-10-2013 22:19</td>
</tr>
<tr>
<td>alm.build.request.vcTag</td>
<td>H_10_Pack_001_b1</td>
</tr>
<tr>
<td>alm.package.name</td>
<td>Pack-001</td>
</tr>
<tr>
<td>alm.project.buildToolType</td>
<td>ANT</td>
</tr>
<tr>
<td>alm.project.buildToolType.keyword</td>
<td>ANT</td>
</tr>
<tr>
<td>alm.project.description</td>
<td>Demo z/OS Project</td>
</tr>
<tr>
<td>alm.project.name</td>
<td>ZOSDEMO</td>
</tr>
<tr>
<td>alm.project.vcName</td>
<td>SVNRepos</td>
</tr>
<tr>
<td>alm.project.vcProjectName</td>
<td>DemoMV5</td>
</tr>
<tr>
<td>alm.project.streamBuildRefV</td>
<td>1</td>
</tr>
<tr>
<td>alm.project.streamType</td>
<td>H</td>
</tr>
<tr>
<td>ftp.password</td>
<td>192.168.253.168</td>
</tr>
<tr>
<td>ftp.userid</td>
<td>ACDMST</td>
</tr>
<tr>
<td>source</td>
<td>D:/ikan/ALM_environments/contbuild/source/382/DemoMV5</td>
</tr>
<tr>
<td>sourceRoot</td>
<td>D:/ikan/ALM_environments/contbuild/source/382</td>
</tr>
<tr>
<td>target</td>
<td>D:/ikan/ALM_environments/contbuild/target/382</td>
</tr>
</tbody>
</table>
Integrating IKAN ALM and Mainframes

**IKAN ALM Report sample, Project stream overview**

**Level Request for project DEMOZOS Base ended with status WARNING**

**Project Information:**
- Name: DEMOZOS
- Description: Project DEMOZOS stored in Subversion for ZOS
- VCR: SVNRepository
- VCR project: DEMOZOS

**Project Stream Information:**
- Head: yes
- Build Prefix: Base

**Level Request Information:**
- OID: 1352
- Description: test scripts
- VCR Tag: Base_b1
- Created by: Global Administrator
- Created on: 2012-08-08 19:01:36.0
- Sent to User Group: ALM Project
- Requested for: 2012-08-08 19:01:36.0
- Action: Request Build
- Target Level: BASE_BUILD

Email received after successful Level Request
What happens behind the scene?

The following z/OS screens show the corresponding actions on the mainframe.

The first z/OS Phase copies the copybook(s) and the programs (and, if existing, also the special components for compiling) to the z/OS environment.

The following screen shows the files that have been collected from the VCR and that are transferred via FTP to the mainframe in a PDS with IKAN ALM.DEMOS.TEST.SRCBATCH as PDS name.
After this FTP Phase, the second phase, Z/OS program compile, is executed.

This phase creates the JCL (See Appendix V: Sample of z/OS compilation JCL), transfers that JCL via FTP to the mainframe and submits the job.

The results will also be available in the IKAN ALM Phase log. That log will list all executed events, step by step, and will tell if the compile has been executed successfully or not.

The following screen shows the JCL jobs that have been submitted and that are executed.

The following screen shows the result of the execution on the mainframe:
When the Job is completed with success, the listing and the load module are transferred via FTP to the IKAN ALM target environment. The following screen shows the sequential file generated by the Xmit step for transferring the load module to the IKAN ALM target environment.

![Sequential Xmit file](image1)

As a result, IKAN ALM has the listing and the load module in his archive. At this point, IKAN ALM is in full control of the remainder of the steps in the lifecycle.

![Content of Build Archive](image2)
From the IKAN ALM archive, the load module can be deployed or promoted to a test or production level by simple starting a Deploy (promote) Level Request that executes a receive step.

The next step is the Deploy or Promote of the compile results. The following screen starts the Deploy.

For the Deploy, IKAN ALM is using the same process as for a Compile/Build: A Deploy Level and Environment with its related z/OS phases must be created.

The z/OS phases defined here are:

- The promote (FTP) of load-modules and other components to the mainframe
- Delete obsolete files and associated components (such as load modules, listings)
- The DB2 Bind statements, transfer (FTP) and DB2 Job execution
- The activation of the CICS load-modules

During this Level Request, the files are copied via FTP to their respective PDS(s) and special jobs can be created and executed on z/OS The Load-modules are received through a transmitted sequential file to the PDS.

The FTP and Job results are analyzed for validating the executed deployment actions. In case of errors, messages are transmitted to the IKAN ALM log and the deployment is stopped.
### Integrating IKAN ALM and Mainframes

#### IKAN ALM log

<table>
<thead>
<tr>
<th>Phase Log</th>
<th>Phase Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Date/Time: 2023-10-16 21:20:00.0</td>
<td>End Date/Time: 2023-10-16 21:20:00.0</td>
</tr>
<tr>
<td>Duration: 00:00:07</td>
<td>Status: SUCCESSFUL</td>
</tr>
<tr>
<td>Message:</td>
<td>IKAN ALM log</td>
</tr>
</tbody>
</table>

#### Mainframe Log

<table>
<thead>
<tr>
<th>Phase Log</th>
<th>Phase Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Date/Time: 2023-10-16 21:20:00.0</td>
<td>End Date/Time: 2023-10-16 21:20:00.0</td>
</tr>
<tr>
<td>Duration: 00:00:07</td>
<td>Status: SUCCESSFUL</td>
</tr>
<tr>
<td>Message:</td>
<td>IKAN ALM log</td>
</tr>
</tbody>
</table>

#### Mainframe Log

<table>
<thead>
<tr>
<th>Phase Log</th>
<th>Phase Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Date/Time: 2023-10-16 21:20:00.0</td>
<td>End Date/Time: 2023-10-16 21:20:00.0</td>
</tr>
<tr>
<td>Duration: 00:00:07</td>
<td>Status: SUCCESSFUL</td>
</tr>
<tr>
<td>Message:</td>
<td>IKAN ALM log</td>
</tr>
</tbody>
</table>

#### Mainframe Log

<table>
<thead>
<tr>
<th>Phase Log</th>
<th>Phase Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Date/Time: 2023-10-16 21:20:00.0</td>
<td>End Date/Time: 2023-10-16 21:20:00.0</td>
</tr>
<tr>
<td>Duration: 00:00:07</td>
<td>Status: SUCCESSFUL</td>
</tr>
<tr>
<td>Message:</td>
<td>IKAN ALM log</td>
</tr>
</tbody>
</table>
The following screen shows the JCL for receiving this transmitted sequential file.

The following screen shows the load modules after FTP from the IKAN ALM archive to the mainframe in a PDS:
At this moment our programs are available for testing in the z/OS Test environment.

The deployment to another z/OS Environment, be it another LPAR or Production environment, is a similar process.

**IKAN ALM –Administrator’s Point of View**

Before Users can start working in IKAN ALM, the IKAN ALM Administrator needs to set up and configure IKAN ALM. Next, he will take care of creating the IKAN ALM projects and specifying the required parameters for the environments and phases.

To make his task as easy as possible, IKAN ALM has introduced the concept of Phases. Phases allow the IKAN ALM Administrator to customize the workflow of the projects by using highly reusable building blocks. Phases can be shared between different Projects, but also between different IKAN ALM installations.

He will use and customize the IKAN ALM pre-defined “Core” phases to transfer the required components to the mainframe, to create the necessary JCL, to submit the JCL and to transfer the results back to IKAN ALM. If needed, he can also create his own “Custom” Phases.

### Step 1

**Step1: Create the global Phases**

**The Phase concept**

To compile or deploy programs one or more phases are executed via IKAN ALM.

In this section, we will first explain the different components of the phases and, next, we will show how a phase is represented and used in IKAN ALM.

**The IKAN ALM Phase Structure**

<table>
<thead>
<tr>
<th>File</th>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>common</td>
<td>20/09/2013</td>
<td>11:50</td>
</tr>
<tr>
<td>models</td>
<td>23/09/2013</td>
<td>18:38</td>
</tr>
<tr>
<td>resources</td>
<td>21/08/2013</td>
<td>10:02</td>
</tr>
<tr>
<td>zosCompilation.xml</td>
<td>08/10/2013</td>
<td>10:06</td>
</tr>
</tbody>
</table>

**The Common Files**

<table>
<thead>
<tr>
<th>File</th>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>getZosProgramProperties.xml</td>
<td>08/10/2013</td>
<td>14:47</td>
</tr>
<tr>
<td>initZosInfo.xml</td>
<td>08/10/2013</td>
<td>14:47</td>
</tr>
<tr>
<td>jchk000.exe</td>
<td>08/10/2013</td>
<td>14:47</td>
</tr>
<tr>
<td>jchk010.exe</td>
<td>08/10/2013</td>
<td>14:47</td>
</tr>
<tr>
<td>linkEditSysin.xml</td>
<td>08/10/2013</td>
<td>14:47</td>
</tr>
<tr>
<td>loadProperties.xml</td>
<td>08/10/2013</td>
<td>14:47</td>
</tr>
<tr>
<td>specialProps.xml</td>
<td>08/10/2013</td>
<td>14:47</td>
</tr>
<tr>
<td>zosActionsFTP.xml</td>
<td>08/10/2013</td>
<td>14:47</td>
</tr>
</tbody>
</table>
As an example, we have here a common script file that is used for linking a COBOL program. This common file will be used as a template to finally generate the correct and complete JCL step for linking a program.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Standard options for linkedit SYSLIN parameters -->
-<project basedir=".*" default="linkeditSysslin">  
  <!-- Completing SYSLIN cards depending on object type and PGM options -->  
  <target name="linkeditSysslin">  
    <echo file="$(syslinFile)" /> // DD */ echo>
      -<if>
        <property name="include.syslib1"/>
        <then>
          <echo file="$(syslinFile)" append="true" > $(include.syslib1) </echo>
        </then>
      </if>
      -<if>
        <property name="include.syslib2"/>
        <then>
          <echo file="$(syslinFile)" append="true" > $(include.syslib2) </echo>
        </then>
      </if>
      -<if>
        <property name="include.syslib3"/>
        <then>
          <echo file="$(syslinFile)" append="true" > $(include.syslib3) </echo>
        </then>
      </if>
    </target>
  </project>
</xml>
```

**The Resource Files**

Resource files are used to define specific, reusable properties

<table>
<thead>
<tr>
<th>Resource File</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration.properties</td>
<td>08/10/2013 14:47</td>
</tr>
<tr>
<td>defaultPgms.properties</td>
<td>08/10/2013 14:47</td>
</tr>
<tr>
<td>environment.properties</td>
<td>08/10/2013 14:47</td>
</tr>
<tr>
<td>extensions.properties</td>
<td>08/10/2013 14:47</td>
</tr>
<tr>
<td>languages.properties</td>
<td>08/10/2013 14:47</td>
</tr>
<tr>
<td>osfamily.properties</td>
<td>08/10/2013 14:47</td>
</tr>
<tr>
<td>parmsFTP.properties</td>
<td>08/10/2013 14:47</td>
</tr>
</tbody>
</table>
As an example, the COBOL2 language definitions from the languages.properties file:

```properties
# Properties for ZOS Languages
# - called after (Environment).properties
# - property format: (language).parameter
# ----------------------------------------------------
# env.prefix=
# env.qualifa=
# COBOL2 programming parameters
COBOL2.program=IGYCRCTL
COBOL2.parms=LIST,LIB,NOSEQ,NOCMPR2,MAP
COBOL2.parmlib=${env.prefix}.${env.qualifa}.PARMLIB
COBOL2.prefix=SYS1.CEE
COBOL2.loadlib=${COBOL2.prefix}.SIGYCOMP
COBOL2.suffix=SCEELKED
COBOL2.cics.parms=LIST,MAP,XREF,NCAL
COBOL2.db2.program=DSNHPC
COBOL2.db2.parms=HOST(COB2),APOST
COBOL2.db2.linkModule=DSNCLI
COBOL2.db2.dcm.program=DBXMMMPR
COBOL2.db2.dcm.parms=DBOPTBAC
COBOL2.db2.dcm.parms.cics=DBOPTCIC
COBOL2.db2.dcm.linkModule=DBXHVPR
COBOL2.idms.parms=(COBOL)
COBOL2.idms.linkModule=IDMSCBL
COBOL2.ims.parms=DSNIPC
COBOL2.ims.parms=(COBOL2)
COBOL2.ims.linkModule=DSNILI
COBOL2.linkedit.program=HEWL
COBOL2.ndvr.type=COBOL
```

The example below shows the generated properties to use the COBOL2 language definitions in a COBOL program:

```properties
#Ant properties
#Mon Sep 30 10:57:09 CEST 2013
pgm.amode=31
pgm.cics=true
pgm.compile.parms=DATA(31),${RENT}
pgm.compilerType=IBM
pgm.db2=true
```
The Model files

Model files are used as templates for JCL steps.
As an example, we added a model for a JCL to compile a COBOL program, with a COBOL2 compiler.
The compileCobol_jcl.model

 crafting a JCL model for COBOL compile step
An IKAN ALM phase and its usage: the z/OS compile phase
The figure below shows the z/OS compile phase.

The objective of the z/OS compile Phase is to compile z/OS programs with, mainly, Assembler, COBOL and PL/1, BMS map languages, and working with CICS and Databases as DB2, Datacom, IDMS or IMS. The Phase will also control the results of the JCL submit and it will collect all files generated by the compile Jobs. Also, when applicable, the DB2 Bind Files will be generated.

This phase assumes that the files for compiling sources and the source program files have already been transferred to the mainframe in the correct PDSs. Normally, this would be done by a dedicated phase.

It is the task of the IKAN ALM Administrator to make sure that the default values for the parameters are set to the company standards. He can easily do that by changing the parameter values in the IKAN ALM web interface.
The execution of the z/OS Compile phase will use the z/OS compile script to finally generate a complete JCL, taking into account all JCL steps to be executed.

The figure below shows the generated JOB card and the STEP card to compile the COBOL program. The complete generated JCL can be found in Appendix V: Sample of z/OS compilation JCL.

```plaintext
...  
<<<<<<<COMPIL2, store object in objlib if compile=ck
<<<<<<<compile listing is stored in IKAN ALM.DEMOS.TEST.LSTALIB
  
<<<<<<<**   SET PARMCOB='LIST,LIB,NOSEQ,NOCMPR2,MAP'
<<<<<<<SET PARMCOB0='DATA(31)'
  
****  
<<<<<<<**  
<<<<<<<**  
>>>>>>>COMPIL  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
<<<<<<<**  
=======
The IKAN ALM Administrator will do this for all the phases required for the mainframe Build (compile) and deploy processes (probably for several z/OS projects).

**An IKAN ALM phase and its usage: the z/OS deployment phase**

Next, we will show you an example of a Deployment Phase. The main phase of the z/OS deployment with IKAN ALM is the z/OS Promotion of components and load-modules' Phase.
Typically, the parameters are very similar, but you can specify another property file for the target environment during the project setup if required.

**Step 2: Create the IKAN ALM project(s)**

Once the global phases have been defined, the IKAN ALM Administrator creates a release-based or package-based mainframe project.

In this example we will show the creation of a package-based mainframe project: ZOSDEMO.

First he needs to define the General settings and the Project Stream (Baseline) settings on the following screen.

During this creation step, IKAN ALM will also automatically create the “Head” Project Stream and the “Base” lifecycle.
Step 3: Adapt the lifecycle (if necessary)

By default, a “Base” lifecycle is created for the Project, which can be used for defining the required Build and Deploy Levels (i.e., the logical environments). If that lifecycle is not sufficient for the project, the IKAN ALM Administrator needs to define a new lifecycle.

The following screen shows how, for example, a Build/Compile level is created.

Once the Build/Compile level is created, it will be displayed on the Life-Cycles Overview screen.
**Step 4: Define the environments and the necessary parameters**

For each (logical) Level (Build, Test or Production), one or more (physical) environments can be defined. The following screen shows the definition of a Build Environment.

Once the Environment is created, the IKAN ALM Administrator can define the necessary parameters for this environment. Examples of parameters for the BUILDZOS environment are: the z/OS FTP Server address, the User ID and Password for connecting with the z/OS LPAR.
Step 5: Add phases

Once the Levels and Environments have been created, the IKAN ALM Administrator can define the Build or Deploy process by adding Phases. When an Environment is being created, IKAN ALM adds, by default, the IKAN ALM core Phases.

For the z/OS platform, the applicable z/OS Phases have to be added in the correct order for execution. Two z/OS Phases need to be inserted: z/OS Copy Sources before Compilation and z/OS programs Compilation.

Once inserted, they need to be put in the right order of execution: first the sources to be compiled have to be transferred to the mainframe and after that the compile process can be started.
**Step 6: Modify the phase parameters**

Each Phase comes with Default values, set by the IKAN ALM Administrator at import.

If required, the default values of these parameters can be modified as shown on the Phase Parameters screen below.

Using the same method, the Deploy Environment is completed with the required z/OS deployment Phases such as: ‘Promote components and load-modules to z/OS’, ‘z/OS Delete Sources and associated files’, ‘z/OS DB2 Binds transfer and activation’, ‘z/OS Cics Load-modules activation’. 

---

**Environment Phase**

<table>
<thead>
<tr>
<th>Phase Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS programs Compilation</td>
<td></td>
</tr>
<tr>
<td>Phase Version</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Fail on Error</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Phase Parameters**

- **alm.phase.builder**: Value = ANT
- **alm.phase.extractBundle**: Value = true
- **alm.phase.mainScript**: Value = zosCompilation.xml
- **dir.zosModels**: Value = None
- **dir.zosProperties**: Value = D:/ikan/ALM_system/phaseProps
- **ftp.active**: Value = None
- **ftp.password**: Value = None
- **propsfile.environment**: Value = None
- **propsfile.extensions**: Value = None
- **propsfile.languages**: Value = $(dir.zosProperties)/languages.properties
- **propsfile.parmsFTP**: Value = None
Next, the IKAN ALM Administrator changes some default parameters for its target environment:

Now that the IKAN ALM Administrator has done his job, the User can start using IKAN ALM for building/compiling, promoting or deploying his programs.
Conclusion

IKAN ALM offers an alternative for pure mainframe-based development by combining an Eclipse-based development environment with a distributed version control repository. On top of that IKAN ALM complements the development process with Application Lifecycle Management and deploy services.

IKAN ALM’s major asset is its concept of Phases. JCL can be very complicated. By using IKAN ALM Phases, you can generate and tailor any JCL step.

Thanks to the phase concept and the available models and resources we can also guarantee an easy and successful implementation (as an average, it will only take a few weeks).

The key element is for you to define your ALM process. Once that has been established, the implementation of IKAN ALM is fast and straightforward.

If you are already using a mainframe solution like CA-Endevor or Serena ChangeMan and you would decide to migrate to IKAN ALM, you will of course need to migrate your CA-Endevor or Serena ChangeMan legacy to IKAN ALM. To do so, we have a standard migration procedure.

In a nutshell: by implementing IKAN ALM, you can continue exploiting the full strengths of your mainframe and seamlessly combine them with new innovative tooling. This will help you cutting down the costs of maintaining different systems, and above all ease the work of your developers as IKAN ALM will take care of the different steps in the lifecycle of your application including its deploy on the mainframe.

For More Information
To know more, visit http://www.ikanalm.com
Contact IKAN Development: info@ikanalm.com

Related Document
Modern Mainframe Development and ALM
The following appendices explain the terminology used by the different ALM mainframe software providers.

**Appendix I: IKAN ALM Terminology**

The following table explains the terms used by IKAN ALM and provides a brief comment for each of them. This will help users of IKAN ALM to have a better understanding of the terminology used.

<table>
<thead>
<tr>
<th>IKAN ALM</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VCR</strong></td>
<td>A Version Control Repository contains the components to manage. Examples of VCRs are: CVS, Subversion, IBM ClearCase, Serena PVCS, Microsoft VisualSourceSafe.</td>
</tr>
<tr>
<td><strong>Project</strong></td>
<td>A tailored Lifecycle process including development, testing, quality assurance and production can be easily defined, implemented and enforced, offering a comprehensive framework across all major platforms including Windows, UNIX, Linux and IBM mainframe systems. IKAN ALM also supports a stream-based project model allowing project managers to easily add Lifecycles to each version of a project, which makes it easy to differentiate between maintenance, “urgency fix” or release build and deploy processes.</td>
</tr>
<tr>
<td><strong>Lifecycle</strong></td>
<td>Defines the Lifecycle(s) from Development to Production Levels for Streams.</td>
</tr>
<tr>
<td><strong>Project Stream</strong></td>
<td>Each IKAN ALM Project contains exactly one HEAD Project Stream and may contain one or more Branches. A Project Stream is a working entity within IKAN ALM</td>
</tr>
<tr>
<td><strong>Level</strong></td>
<td>Defines every step of the Lifecycle from Development to Production, supporting physical Environments.</td>
</tr>
<tr>
<td><strong>Environment</strong></td>
<td>IKAN ALM uses the (logical) Level concept in which (Build/Deploy) environments can be defined. Every environment represents a Machine (Server/OS) on the network where the Source and Target Locations are defined for executing Phases. This is a unique architectural IKAN ALM feature, representing the true multi-platform aspect of IKAN ALM.</td>
</tr>
<tr>
<td><strong>Package</strong></td>
<td>Instead of using a Release Project, IKAN ALM may work with Packages in the Project. The required files must be added manually from the VCR to the Package IKAN ALM does not contain the Sources. It only knows the link to the files in the VCR project.</td>
</tr>
<tr>
<td><strong>Level Request</strong></td>
<td>A Level Request in IKAN ALM starts a Build, Deploy and Rollback in the Environment(s).</td>
</tr>
<tr>
<td><strong>Build Request</strong></td>
<td>The Build Level Request type in IKAN ALM will usually take care of a compile procedure for components.</td>
</tr>
<tr>
<td><strong>Phase</strong></td>
<td>users can extensively customize the workflow of their projects, by using highly reusable building blocks, called Phases. By using the import/export features, Phases can not only be shared between different Projects, but also between different IKAN ALM installations.</td>
</tr>
<tr>
<td><strong>Script (Ant)</strong></td>
<td>Runs the process (e.g., for build, compilation, deployment, copy, etc.) using the Source and Target locations on the Agent machine. A script can use property files, models and other scripts generally defined in a Phase.</td>
</tr>
<tr>
<td><strong>Build#</strong></td>
<td>IKAN ALM generates a unique build number that can be used in several processes to identify the output from the (build/compile) procedures. IKAN ALM is also able to put this information on members in a Partitioned Dataset on z/OS.</td>
</tr>
</tbody>
</table>
Integrating IKAN ALM and Mainframes

Approval
IKAN ALM allows setting up a hierarchy in the Approval Groups. For example, Group2 may only approve if Group1 has approved first. Groups are based on the Enterprise Security System users.

Rollback
In IKAN ALM, an automatic rollback can be executed for every kind of output, which will allow the customer to completely automate a rollback operation. Typically, it runs a previous version of your choice.

Machine
A machine runs an IKAN ALM Agent which will take care of building/deploying the software components. Linux/UNIX (flavors) and Windows platforms. The Agent (LUW) machine can update more than one LPAR via FTP connections. Z/OS Phases might be reused with models and PDS definitions using several FTP connections.

Release Number/Incident Number
IKAN ALM has an ITS-plugin that allows you to easily link existing issue or defect tracking systems. Issues are accumulated along the Lifecycle and updated automatically.

Archive
IKAN ALM compresses and saves all Build results in Archives and keeps them in a dedicated location. Archives are identified with the Build Tag.

Report
IKAN ALM has a web interface to view per project and level request what happened. On top of that, an ALM-Reports tool allows creating more Reports about Global and Project Administration and Package activities.

Phase adds Remarks
Extension/Object-type
The extension/objtype determines the processing needed for a certain file type. This is defined by a property file and scripts. Object-types are used for z/OS activities.

Obsolete File
IKAN ALM has no process for scratching individual files. This action is resolved with an Environment Phase which scratches the source component using the “.to_be_deleted” suffix in the VCR. Associated z/OS components are deleted in their PDS.

Appendix II: CA-ENDEVOR Terminology

The following table maps the terms used by IKAN ALM and CA-Endevor and provides a brief comment for each of them. This will help the respective users of IKAN ALM or CA-Endevor to have a better understanding of the terminology used.

<table>
<thead>
<tr>
<th>IKAN ALM</th>
<th>CA Endevor</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCR</td>
<td>Database/Delta</td>
<td>CA-Endevor assumes VCR versions with the Image and Delta file(s).</td>
</tr>
<tr>
<td>Project</td>
<td>System and/or</td>
<td>Within IKAN ALM, the defined project needs attributes to tell IKAN ALM to which CA-Endevor System/Sub-system the Software Items should be added in Environment parameters.</td>
</tr>
<tr>
<td></td>
<td>Sub-system</td>
<td></td>
</tr>
<tr>
<td>Lifecycle</td>
<td>Map</td>
<td>Defines the Lifecycle from Development to Production.</td>
</tr>
<tr>
<td>Stream</td>
<td>Not available</td>
<td>CA-Endevor works with a unique Project version.</td>
</tr>
<tr>
<td>Package</td>
<td>Package</td>
<td>CA-Endevor groups components in Batch packages.</td>
</tr>
<tr>
<td>Level</td>
<td>Stage</td>
<td>Defines every step of the Lifecycle from Development to Production.</td>
</tr>
<tr>
<td>Environment</td>
<td>(Stage)</td>
<td>This is a unique architectural IKAN ALM feature, not known in CA-Endevor, representing the true multi-platform aspect of IKAN ALM.</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Level Request</td>
<td>(Move) Action</td>
<td>CA-Endevor distinguishes more actions, but they are not directly applicable to IKAN ALM. For example, to delete a component from the Production environment, the components should be deleted from the VCR; the project should be built and deployed, and tested in all the Levels between Development and Production, ensuring that this deletion does not jeopardize the Production. This delete task may be activated with a SVN property on the component (don’t delete). Next, IKAN ALM may assume the deletion during the deployment by using the SVN property in a script.</td>
</tr>
<tr>
<td>Build Request</td>
<td>(Add) Action</td>
<td>The Build Level Request in IKAN ALM will usually take care of populating CA-Endevor with the Software Components (ADD action).</td>
</tr>
<tr>
<td>Phase</td>
<td>Processor group</td>
<td>The processor group in CA-Endevor determines the ultimate process to run within a certain type. For example, the Processor Type COBOL might have processor groups for COBOL, DB2, CICS, BATCH, IMS etc.</td>
</tr>
<tr>
<td>Script (Ant)</td>
<td>Processor</td>
<td>Runs the process (e.g., for build or compilation).</td>
</tr>
<tr>
<td>Idrdata/Build#</td>
<td>Footprint</td>
<td>Build# or Build number: is an incremental number given after each software build. IDR DATA: Identification record data field. Identification records have a fixed format and fixed content, both defined by the program management binder. Is used by IBM Endevor footprints contain the following information: site ID, environment, system, subsystem, element, type, stage, version/level, and generate date/time.</td>
</tr>
<tr>
<td>Approval</td>
<td>Approval</td>
<td>CA-Endevor allows defining several Approval Groups which are in the same hierarchy. Every group may approve on any moment.</td>
</tr>
<tr>
<td>Rollback</td>
<td>Backout</td>
<td>CA-Endevor allows reversing the result from a promotion/delivery if it is a member(s) in a Partitioned Dataset (PDS). In the case of DB2 a (manual) rebind should be executed.</td>
</tr>
<tr>
<td>Machine</td>
<td>Ship</td>
<td>CA-Endevor only supports other z/OS Logical Partitions (LPARS).</td>
</tr>
<tr>
<td>Release Number/Incident Number</td>
<td>CCID</td>
<td>The release/incident number within IKAN ALM may be passed to CA-Endevor as the CCID (Change Control Identifiers) most often correspond to mechanisms such as work order requests or request-for-service numbers.</td>
</tr>
<tr>
<td>Archive</td>
<td>Not available</td>
<td>CA-Endevor keeps these components in Stage Level with CCID’s.</td>
</tr>
<tr>
<td>Extension/Object-type</td>
<td>Type</td>
<td>The extension/objtype determines the processing needed for a certain type.</td>
</tr>
</tbody>
</table>
Appendix III: Serena ChangeMan ZMF terminology

The following table maps the terms used by IKAN ALM and Serena ChangeMan ZMF and provides a brief comment for each of them. This will help the respective users of IKAN ALM or ChangeMan to have a better understanding of the terminology used.

<table>
<thead>
<tr>
<th>IKAN ALM</th>
<th>ChangeMan ZMF</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCR</td>
<td>Baseline/Delta/Package</td>
<td>ChangeMan assumes VCR functionalities as Check-Out, Commit (Baseline Ripple), Check-In (Freeze) from the Package Lifecycle.</td>
</tr>
<tr>
<td>Project</td>
<td>Application</td>
<td>ChangeMan has the same concepts as IKAN ALM but only for IBM mainframe systems. Also the stream-based project is not available.</td>
</tr>
<tr>
<td>Lifecycle</td>
<td>Stage/ Promotion Levels</td>
<td>Defines the Lifecycle from Development to Production.</td>
</tr>
<tr>
<td>Stream</td>
<td>Not available</td>
<td>ChangeMan works with a unique Project version.</td>
</tr>
<tr>
<td>Package</td>
<td>Package</td>
<td>ChangeMan uses the Package for the Development process up to the Stage action. IKAN ALM leaves development actions to the customer IDE and the VCR. Both IKAN ALM with Level Requests and ChangeMan with Staging, manage Build (Compile) requests, as well as Deployments with the Approval supervision for the Package. ChangeMan, however, needs to update the Baseline &amp; Stacked Reverse Delta supports in double with the Production and the Package.</td>
</tr>
<tr>
<td>Level</td>
<td>Promotion Level</td>
<td>Defines every Level of the Lifecycle from Development to Production.</td>
</tr>
<tr>
<td>Environment</td>
<td>Site (Local or Remote)</td>
<td>The Local or Remote Site concept in ChangeMan is covered by the IKAN ALM environment concept. An IKAN ALM level (a logical step) can have one or more environments.</td>
</tr>
<tr>
<td>Phase</td>
<td>Procedures or skeletons</td>
<td>The skeletons in ChangeMan determine the process to run within a certain type. For example, the Procedure CMNCOB2 might have process skeletons for COBOL, DB2, CICS, IMS, etc.. Depending on the Source options.</td>
</tr>
<tr>
<td>Level Request</td>
<td>Promotion/ Demotion</td>
<td>ChangeMan knows more actions, but they are not directly applicable to IKAN ALM. For example, to delete a component from the Production environment, the components should be renamed into the VCR with the special “.to_be_deleted” suffix. Next, the project should be built and deployed, and tested in all the Levels between Development and Production, ensuring that this deletion does not jeopardize the Production. This delete task will be activated with the suffix of the component (don’t delete). Next IKAN ALM may assume the deletion during the deployment by using the suffix in the dedicated Phase.</td>
</tr>
<tr>
<td>Build Request</td>
<td>(ST) Action</td>
<td>ChangeMan takes care of the compile procedure the same way as IKAN ALM.</td>
</tr>
<tr>
<td>Script (Ant)</td>
<td>Skeleton Procedure</td>
<td>Runs the process (e.g., for build or compilation or deploy).</td>
</tr>
<tr>
<td>Build#</td>
<td>Package Number</td>
<td>ChangeMan uses the Package number for versioning files.</td>
</tr>
<tr>
<td>Approval</td>
<td>Approval</td>
<td>ChangeMan allows defining several Approval Groups which are hierarchical. Every group may approve one after the other.</td>
</tr>
</tbody>
</table>
Integrating IKAN ALM and Mainframes

Machine Site ChangeMan only supports other z/OS Logical Partitions (LPARS). The ChangeMan site is the Local or a Remote LPAR.

Release Number/Incident Number Not available ChangeMan does not use Incident numbers. In the Package description panel, a reason may be entered for all included components.

Archive Package ChangeMan contains components in the Package which is frozen before the deployment. It designs the version to deploy.


Obsolete File Scratch/Rename ChangeMan assumes the Scratch and the Rename functionalities during the Promote. Will be supported through a custom phase.

Object-type Library type The objtype determines the processing needed for a certain type. IKAN ALM can use the same codes.

IKAN Impact Analysis Tool Impact-Analysis ChangeMan Impact Analysis covers Source, Copy, JCL, Proc and DSN names relationships. The Impact Analysis solution from IKAN ALM, permits you to create an Impact Analysis table, based on the information available in the Version Control Repository. IKANALM reports are available to query that Impact Analysis table and as such you get the same and more results as with the Change Man Impact Analysis solution.

Analysis Tool

Not available Merge & Reconcile IKAN ALM does not need to support this because it is a task of the VCR.

As Archive Freeze In relation to this ChangeMan concept, IKAN ALM creates an Archive at the end of every Build containing all components to deploy. It is this Archive that used for the next.

Not available Baseline It is a ChangeMan concept that duplicates (or not) the Production Level used for future package developments considering it is the version 0 as Reference in Production. IKAN ALM doesn’t assume this concept because it is the VCR task to define the versions of components. IKAN ALM creates or presents the Tag for a Build version.

Appendix IV: Available z/OS IKAN ALM Phases

The following table maps the Phases used by IKAN ALM for compiling and deploying components to Mainframe Environments. Note that for IDMS, a Phase collects the dictionary components and the next phase installs them into another one.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS copy Source to Target</td>
<td>Build/Deploy</td>
<td>A dedicated Phase for copying the z/OS Components (Sources or Objects) to the IKAN ALM Target Environment. This Phase only transfers selected component types.</td>
</tr>
<tr>
<td>z/OS copy Sources before Compilation</td>
<td>Build</td>
<td>This Phase transfers via FTP Copybooks, Linkedit Control Cards (LCT cards) Assembler, COBOL PL/1 Programs and BMS/SDF2 Maps to PDS(ies) in the Mainframe Environment.</td>
</tr>
<tr>
<td>Component Description</td>
<td>Deployment Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>z/OS Programs Compilation</strong></td>
<td>Build</td>
<td>For each Map (firstly) and each Program (secondly), the Phase generates a compile JCL depending on the Source contents and language using included JCL models. Next, each JCL is executed by JES under FTP and the resulting Job is analyzed to know its status. Next, the generated compile Listing, Load-module and DB2 DBRM, Datacom Plan are transferred to the IKAN ALM Target Environment. Optionally, DB2 Binds may be generated from models. Note that for CA-Endevor the Repository will be updated for compiling with it.</td>
</tr>
<tr>
<td><strong>z/OS Promotion of components and load-modules</strong></td>
<td>Deploy</td>
<td>z/OS components in the IKAN ALM archive are transferred to their PDS(s) of the Mainframe Environment. Exception: the Load-modules which are transferred to flat files before a generated JCL using included JCL models is executed by JES under FTP for receiving them in their PDS(s).</td>
</tr>
<tr>
<td><strong>z/OS Delete Sources and associated objects</strong></td>
<td>Deploy</td>
<td>All Sources identified by the “to_be_deleted” suffix are deleted in PDS(s) of the Mainframe Environment by FTP. Also, the associated Listings, Load-modules, DBRMs, Plans and DB2 Binds are deleted in their PDS(s). No action in DB2 and Datacom Databases.</td>
</tr>
<tr>
<td><strong>z/OS DB2 Binds transfers and activation</strong></td>
<td>Deploy</td>
<td>If DB2 is used, Bind files are copied to their PDS(s) and a JCL is generated using included JCL models and executed by JES under FTP for running these Binds on the DB2 Database.</td>
</tr>
<tr>
<td><strong>z/OS CICS Load-modules activation</strong></td>
<td>Deploy</td>
<td>If there are CICS Maps or Programs, a JCL is generated using included JCL models, and executed by JES under FTP for running the PHASEIN commands on a CICS.</td>
</tr>
<tr>
<td><strong>z/OS Update Datacom components Promotion</strong></td>
<td>Deploy</td>
<td>If there are Plans, the Phase generates a JCL using included JCL models and executed by JES under FTP for importing Plans on the Datacom Database.</td>
</tr>
<tr>
<td><strong>z/OS Update Endevor components Promotion</strong></td>
<td>Deploy</td>
<td>If the CA-Endevor Repository is active on the Mainframe, the Phase generates a JCL using included JCL models and executed by JES under FTP for moving components from the Stage ID to the corresponding Level.</td>
</tr>
<tr>
<td><strong>z/OS SQL DB2 updates Execution</strong></td>
<td>Deploy</td>
<td>If DB2 is used, DDL and SQL statements may be applied with variable substitutions as owner, qualifier. After the transfer of DDL and SQL commands concatenated into 2 members, 2 JCLs are generated using included JCL models and executed by JES under FTP for running DDL and, next, SQL on the DB2 Database.</td>
</tr>
<tr>
<td><strong>z/OS Update Debugger</strong></td>
<td>Deploy</td>
<td>For instance, for the Xpediter tool, this phase copies Xpediter components from a FILEIO file to another FILEIO using the components list in the IKAN ALM target environment.</td>
</tr>
<tr>
<td><strong>z/OS Copy Pds Members</strong></td>
<td>Deploy</td>
<td>This phase transfers components from PDS(s) of a z/OS environment to PDS(s) of another z/OS environment using the components list in the IKAN ALM target environment.</td>
</tr>
<tr>
<td><strong>z/OS Update QMF</strong></td>
<td>Deploy</td>
<td>This phase imports QMF components to a QMF DB2 sub-system using the components list in the IKAN ALM target environment.</td>
</tr>
<tr>
<td><strong>z/OS Collect IDMS components</strong></td>
<td>Build</td>
<td>For the first build, this Phase generates a JCL using included JCL models that is executed in the Mainframe Environment by JES under FTP. This one collects IDMS components and info about date and parent relations in the IDD of development. For the rebuild before deployment, the Phase controls the correlation with the target IDD with execution of another generated JCL.</td>
</tr>
</tbody>
</table>
### Integrating IKAN ALM and Mainframes

<table>
<thead>
<tr>
<th>z/OS IDMS components Promotion</th>
<th>Deploy</th>
</tr>
</thead>
<tbody>
<tr>
<td>This phase transfers components to temporary files in the Mainframe Environment and she generates a JCL using included JCL models that is executed by JES under FTP for updating the target IDD.</td>
<td></td>
</tr>
</tbody>
</table>

... (on demand)

### Appendix V: Migration to IKAN ALM

Before you can work with IKAN ALM, components must be installed in a VCR. This is a big difference with CMN where the full VCR is included in Packages, Baseline and SRDeltas PDSs, or with Endevor where components are in workspace PDSs.

IKAN has developed an Ant solution for migrating components from the CMN versioning system to VCR projects (Subversion or Clearcase). The Tool supports the collect of versions in the SRDeltas, Baseline and Packages for all types of components based on the CMN project concept. The results are the same versioning levels in the VCR Projects and same Package definitions that you had in CMN.

The solution supports migrations from classic PDS.

IKAN also has developed an Ant solution for updating VCR Projects and IKAN ALM Packages from other tools (z/OS tools or Windows/Unix tools) to automatically version and deploy some components using the package process.

### Appendix VI: Sample of z/OS compilation JCL

The following JCL is fully generated by the z/OS Compilation Phase used by IKAN ALM for compiling a component into the Mainframe Environment.

```plaintext
//ADCDMSTC JOB (5145,00000,2233,T),,'IKAN',
//          MSGLEVEL=(1,1),MSGCLASS=X,
//          CLASS=A,REGION=8M
//*
//**XEQ ROUTEID=ADCD
//*******************************************************************
//**   COPYING THE PROGRAM IN SOURCE WORK FILE                     **
//*******************************************************************
// SET SRCOMPIL=SOURCE
//COPYSRC EXEC PGM=IEBGENER
//SYSTSIN DD DUMMY
//SYSTSPRT DD SYSOUT=* 
//SYSPRINT DD SYSOUT=* 
//SYSUT1 DD DISP=(SHR),
//       DSN=IKANALM.DEMOS.TEST.SRCBATCH(DEMO21)
//SYSUT2 DD DISP=4,PASS),DSN=%%%SRCOMPIL,
//       UNIT=SYSDA,SPACE=(CYL,(10,10)),
//       DCB=(RECFM=FB,LRECL=80,BLKSIZE=0)
//SYSIN DD DUMMY
//*******************************************************************
//**    COMPILE COBOL2, store object in objlib if compile=ok
```
/** compile listing is stored in IKANALM.DEMOS.TEST.LSTALIB
&view rule:********** ******************
/**
/** SET PARMCOB='LIST,LIB,NOSEQ,NOCMPR2,MAP'
/** SET PARMCOBO='DATA(31)'
/**
/** COPILE THE ELEMENT
/**
/** COBOL EXEC PGM=IGYCRCTL,COND=(4,LT),
/** PARM="/PARMCOB,/PARMCOB"
/** STEPLIB DD DISP=SHR,DSN=SYS1.COB2COMP
/** SYSIN DD DISP=(OLD,PASS),DSN=&&&SRCOMPIL
/** SYSLIN DD DISP=(,PASS),DSN=&&OBJECT,
/** UNIT=SYSDA,SPACE=(CYL,(2,2)),
/** DCB=(RECFM=FB,LRECL=80,BLKSIZE=0)
/** SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(5,3))
/** SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(5,3))
/** SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(5,3))
/** SYSUT4 DD UNIT=SYSDA,SPACE=(CYL,(5,3))
/** SYSUT5 DD UNIT=SYSDA,SPACE=(CYL,(5,3))
/** SYSUT6 DD UNIT=SYSDA,SPACE=(CYL,(5,3))
/** SYSUT7 DD UNIT=SYSDA,SPACE=(CYL,(5,3))
/** SYSPRINT DD DISP=(,PASS),DSN=&&COMPLIST,
/** UNIT=SYSDA,SPACE=(TRK,(10,10),RLSE)
/** DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0)
/** SYSLIB DD DISP=SHR,DSN=IKANALM.DEMOS.TEST.COPYLIB
/** DD DISP=SHR,DSN=IKANALM.DEMOS.INTG.COPYLIB
/** DD DISP=SHR,DSN=IKANALM.DEMOS.QUAL.COPYLIB
/** DD DISP=SHR,DSN=IKANALM.DEMOS.PROD.COPYLIB
/** ALLOCLCT EXEC PGM=IEFBR14
/** SYSPRINT DD SYSOUT=*;
/** LCTFILE DD DISP=(NEW,PASS,DELETE),DSN=&&LCTFILE,
/** UNIT=SYSDA,SPACE=(TRK,(1,1,1)),
/** DCB=(DSORG=PO,RECFM=FB,LRECL=80,BLKSIZE=0)
/** CREATLCT EXEC PGM=IEBGENER
/** SYSPRINT DD SYSOUT=*;
/** SYSUT1 DD *
/** SYSUT2 DD DISP=(MOD,PASS),DSN=&&LCTFILE(DEMO21)
/** SYSIN DD DUMMY
/** COPYLCT EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*  
//INDD00 DD DISP=SHR,DSN=IKANALM.DEMOS.TEST.LCTLIB  
//INDD01 DD DISP=SHR,DSN=IKANALM.DEMOS.QUAL.LCTLIB  
//INDD02 DD DISP=SHR,DSN=IKANALM.DEMOS.PROD.LCTLIB  
//OUTDD1 DD DISP=(MOD,PASS),DSN=&LCTFILE  
//SYSSIN DD *  
COPY OUTDD=OUTDD1  
INDD=INDD00,INDD01,INDD02,INDD03  
SELECT MEMBER=DEMO21  
/  
//******************************************************************  
//** LINKEDIT PROGRAM **  
//******************************************************************  
// SET PARMLNK='LIST,MAP,XREF,NCAL'  
// SET LINKOPT='RENT,AMODE(31),RMODE(ANY),'  
//LKEDT EXEC PGM=HEWL,COND=(4,LT),  
// PARM='&PARMLNK,&LINKOPT'  
//SYSLMOD DD DISP=SHR,DSN=IKANALM.DEMOS.TEST.LOADLIB(DEMO211)  
//SYSEFSD DD DUMMY  
//SYSPRINT DD DISP=(,PASS),DSN=&LINKLIST,  
// UNIT=VIO,SPACE=(TRK,(10,10)),  
// DCB=(RECFM=FBA,LRECL=121,BLKSIZE=0)  
// SYSLIB DD DISP=SHR,DSN=IKANALM.DEMOS.TEST.LOADLIB  
// DD DISP=SHR,DSN=IKANALM.DEMOS.QUAL.LOADLIB  
// DD DISP=SHR,DSN=IKANALM.DEMOS.PROD.LOADLIB  
// SYSLIN DD *  
// IDENTIFY DEMO211('DEMO21/ADCDMST/000003')  
// NAME DEMO211(R)  
//******************************************************************  
//** TRANSMIT PROGRAM **  
//******************************************************************  
//CLEARSEQ EXEC PGM=IEFBR14  
//DD01 DD DISP=(MOD,DELETE),  
// DSN=IKANALM.DEMOS.TEST.DEMO211,
//          UNIT=SYSDA,SPACE=(TRK,(1)),
//          LRECL=80,BLKSIZE=3120,RECFM=FB
//*
//XMITLOAD EXEC PGM=IKJEFT01,COND=(4,LT)
//SYSPRINT DD SYSOUT=*  
//SYSTSIN DD *
//XMIT (ADCD.*) -
//DSNAME(‘IKANALM.DEMOS.TEST.LOADLIB’) MEM(Demo211)-
//OUTDSNAME(‘IKANALM.DEMOS.TEST.DEBO211’) NOLOG NONOTIFY
//*
//PRTCMPA IF (COBOL.RUN EQ TRUE) THEN
//** PRINT THE COMPILE LISTING **
//****************************************************************************
//** PRINT THE COMPILE LISTING **
//****************************************************************************
//PRNTCOMP EXEC PGM=IEBGENER
//SYSTSIN DD DUMMY
//SYSTSPRT DD SYSOUT=*  
//SYSPRINT DD SYSOUT=*  
//SYSUT1 DD DISP=(OLD,PASS),DSN=&&COMPLIST
//SYSUT2 DD SYSOUT=*  
//SYSIN DD DUMMY
//PRTCMPZ ENDIF
//*
//PRTLNKA IF (LKEDT.RUN EQ TRUE) THEN
//** PRINT THE LINKEDIT LISTING **
//****************************************************************************
//** PRINT THE LINKEDIT LISTING **
//****************************************************************************
//PRNTLINK EXEC PGM=IEBGENER
//SYSTSIN DD DUMMY
//SYSTSPRT DD SYSOUT=*  
//SYSPRINT DD SYSOUT=*  
//SYSUT1 DD DISP=(OLD,PASS),DSN=&&LINKLIST
//SYSUT2 DD SYSOUT=*  
//SYSIN DD DUMMY
//****************************************************************************
//** FORMAT THE LINKEDIT LISTING **
//****************************************************************************
//IFSFTLKD IF (NOT ABEND) THEN
//FRMTLKD EXEC PGM=SORT
//SORTSNAP DD  SYSOUT=*  
//SORTWK01 DD DISP=(PASS),UNIT=SYSDA,SPACE=(CYL,(10,10),RLSE)
//SORTWK02 DD DISP=(PASS),UNIT=SYSDA,SPACE=(CYL,(10,10),RLSE)
//SORTWK03 DD DISP=(PASS),UNIT=SYSDA,SPACE=(CYL,(10,10),RLSE)
//SORTWK04 DD DISP=(PASS),UNIT=SYSDA,SPACE=(CYL,(10,10),RLSE)
//SORTIN DD DISP=(OLD,DELETE),DSN=&&LINKLIST
//SORTOUT DD DISP=(NEW,PASS,DELETE),DSN=&&LISTLKD,
//UNIT=SYSDA,SPACE=(CYL,(5,5)),
//DCB=(DSORG=PS,RECFM=FBA,LRECL=133,BLKSIZE=0)
//SYSIN DD *
SORT FIELDS=COPY
OUTREC FIELDS=(1,121,12X)
/*
//SYSOUT DD SYSOUT=* 
//SYSUDUMP DD SYSOUT=* 
//SYSMDUMP DD SYSOUT=* 
//SYSABEND DD SYSOUT=* 
//IFEFTLKD ENDIF
//*******************************************************************
//**    COPY THE LISTINGS                                          **
//*******************************************************************
//IFDEFST1 IF (NOT ABEND) THEN
//LIST EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=* 
//SYSUT2 DD DISP=SHR, // 
// DSN=IKANALM.Demos.Test.LSTALIB(DEMO21) // 
//SYSIN DD DUMMY // 
//SYSUT1 DD DISP=(NEW,DELETE,DELETE),DSN=&&NULLSEQ, // 
// UNIT=SYSDA,SPACE=(TRK,(1,1)), // 
// DCB=(DSORG=PS,RECFM=FBA,LRECL=133,BLKSIZE=0) // 
//* DD DISP=(OLD,DELETE),DSN=&&PCMLIST // 
// DD DISP=(OLD,DELETE),DSN=&&COMPLIST // 
// DD DISP=(OLD,DELETE),DSN=&&LISTLKD // 
//IFDEFST1 ENDIF // 
//*/
//IFDEFST1 IF (RC GT 4 OR ABEND) THEN
//FAILURE EXEC PGM=IEBGENER,MAXRC=0
//SYSPRINT DD SYSOUT=* 
//SYSUT1 DD * 
// JOB 1626 FAILED 
//*/
//SYSUT2 DD SYSOUT=* 
//SYSIN DD DUMMY 
//IFDEFST1 ENDIF