Source Code Delivery for AIP Console

Background and Approach

There are two main approaches to "source code" delivery when onboarding or rescanning an application in AIP Console.

1. **Single Zip file (Preferred)** - This requires as input, a single Zip file containing all relevant "Source Code" artefacts. The structure of the folders within the Zip file needs to also remain consistent from release to release. One way to do this is to create separate folders for each technology and also folders by sub technology within technology. This requires that some initial preparation is put into the creation of this Zip file.
2. **Defined folders and files** - This requires the configuration of all the files and folders with AIP Console, that contains the same files and folders that would be captured in a s**ingle Zip file**.

The "source code" delivery needs to contain more than just source code (please see technology specifics below). To perform system-level analysis, AIP Console requires:

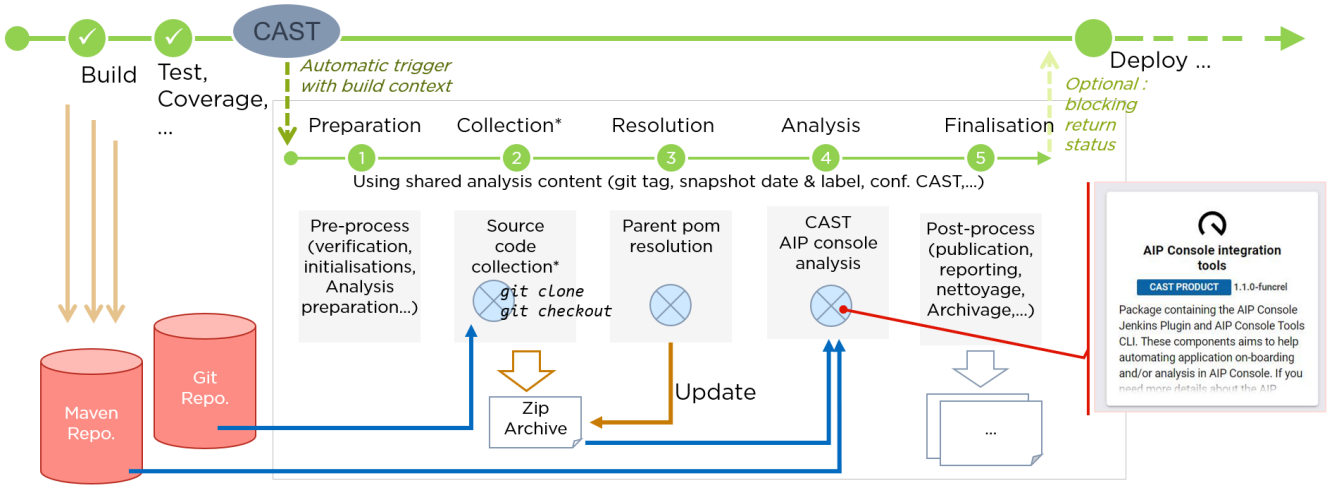
- Source code
- Build scripts. These can be pom.xml files, makefiles and so on
- Libraries. These can be extracted from the Repository Manager like Nexus and so on. An automatic link can be made to Maven if an "m2" is included in the Zip
- Generated source code for inclusion in the compilation
- Generated resources (wsdl, xml, …) for inclusion in the package
- DDL build scripts or DDL extracts. The first is the script used to build databases the second is a DDL extract from an existing database.

Source code repositories often contain more than just the source artefacts required to make up applications. Other files could include application documentation, test code, startup scripts and migration code to name a few. In order to ensure you have the right artefacts, it is key to establish a suitable point within the build, release or test cycle where all the relevant artefacts are available.

Ideally, the Zip file or "source code" folders should be as “clean” as possible, containing only the required source artefacts. Early feedback of alerts from AIP Console and early visibility of dashboards will assist with the process of presenting all the relevant source artefacts.

If an application team uses a staging location for the source artefacts, in order to build, a good approach is to use these same folder locations to create a Zip file that can then be passed to AIP Console for system-level analysis. The frequency of the Zip creation does not have to match the build cadence, especially in toolchains where multiple builds are done on a very frequent basis.

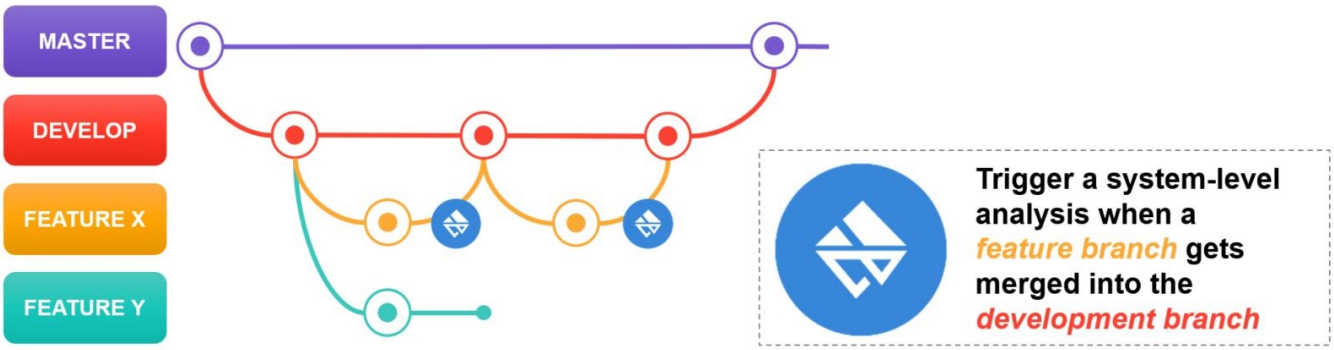
The diagram below shows how the creation of the Zip can be integrated into the existing toolchain



**Source code collection: shall include* *full source code required to CAST analysis: .java, .c#, SQL, as well as data extracted for XXL analysis, or technologies like SAP, People Soft, or generated artifacts)*
In case there is no remote Maven repository, a local ‘m2’ should be included into the zip archive

Linking the right version of the database structure to the right version of source artefacts is also crucial. Ideally, an automated extract of the database definition language (DDL) to match the source artefacts, will ensure consistency.

Frequency of presenting a new Zip file to AIP Console is also important. Performing a system-level analysis after the check-in of each source artefact will maybe result in a few changes, but will also create a burden for developers to consume the results each time. A better approach is to trigger a system-level analysis when a feature branch gets merged into the development branch.



The key aim is to establish an automated and repeatable process which is simple for developers to consume.

AIP Console also has REST API's which will allow the automated presentation of a Zip file to AIP Console and the corresponding processing of results and dashboards. This allows the creation of an unattended rescan process.

Architecture Specifics

There are two main application types

- Monolithic application (self-contained, single build architecture)
- Microservices application (de-coupled, multiple build architecture)

Monolithic applications - Creating a Zip for a monolithic application is normally easier, because of the very nature of the application architecture and the tight coupling of the source artefacts. As described above, creating a Zip of source artefacts from the build process or build staging folders will most likely ensure all the correct artefacts are included.

Microservices applications - Microservice applications are often split across multiple repositories, which often have their own build and release cycles. Building a single Zip file from this type of environment does require more coordination. The application teams may need to create a shared integration environment, where source artefacts are delivered as part of the build and then a zip file is created at an agreed point in time. A best practice approach is to create one folder per microservice with these folders then being Zipped together for a single Zip file for AIP Console. For additional reference please follow this link.

Technology Specifics

When onboarding an Application that includes .NET based source code, you must tell the AIP Console where to find the .NET framework (system assemblies). The location of these assemblies is crucial to ensure that the analysis of the .NET Application completes successfully. See Configuring source code delivery as Zip file for .NET.

When onboarding an application that is built using Maven there are 2 options

1. If AIP console has access to the remote maven repository, there is no need to include the jars in the zip, AIP console will download the jars automatically
2. If AIP console cannot access the remote maven repository, there is a need to include the jars with the maven folder structure in the zip

See Configuring source code delivery as Zip file for Maven

If the application to be onboarded uses a database, information on how to extract the database definition can be found at this link

Technology specific information can be found at this link

Sample Folder Structures

Below is an example structure and tree that could be used starting from the SRC folder, as indicated below and differentiated according to the type of technology used in the project:

- **SRC**
  - *Specific Trees for Technologies*

If the Project is multi-technology, the convention must be adopted to add a postfix to the SRC folder as follows:

- **SRC-*Technology-1***
  - *Specific Trees for Technology-1*
- **SRC-*Technology-2***
  - *Specific Trees for Technology -2*
- **SRC-*Technology-n***
  - *Specific Trees for Technology -n*

Example:

- SRC-*J2EE*
  - *Specific Trees for J2EE*
- SRC-*CPP*
  - *Specific Trees for CPP*
- SRC-*TibcoBW*
  - *Specific Trees for TibcoBW*

Where the technology requires it, all the libraries used must be inserted in the following folders:

- **Internal_LIB**
  - *Specific Trees for Technologies*
- **ThirdParts_LIB**
  - *Specific Trees for Technologies*

Note:

In case the Project is multi-technology the above rule should be followed as for the SRC folders.

- **Internal_LIB- *Tecnologia-n***
  - *Specific Trees for technology-n*
- **ThirdParts_LIB- *Tecnologia-n***
  - *Specific Trees for technology-n*

Below are the technologies and in detail with the specific tree and the types of files that must be inserted.

For C and CPP Projects

An example of a structure is shown below:

- **SRC**
  - *C-CPP*
  - *Include*
  - *MacroDef*
  - *PrecompiledH*
  - *EnvProfiles*
  - *STLHeader*

It is specified that within the indicated folders it is possible to insert further folders according to the version of the compiler in use.

For Mainframe Projects

An example of a structure is shown below::

- **SRC**
  - *COB*
  - *CPY*
  - *JCL-JOB*
  - *JCL-PRC*
  - *IMS-PSB*
  - *IMS-DBD*
  - *IMS-DC*
  - *CICS-BMSCICS-CSD*

A typical source structure in the Mainframe environment is given as an example:

Type	Folder	Files extension
COBOL program	COB	*.COB o *.CBL
COBOL copybook	CPY	*.CPY o *.COP
JCL job	JCL-JOB	*.JCL
JCL procedure	JCL-PRC	*.PRC
IMS PSB	IMS-PSB	*.PSB
IMS DBD	IMS-DBD	*.DBD
IMS-DC	IMS-DC	*.TRA
CICS Mapset	CICS-BMS	*.BMS
CICS CSD flat file	CICS-CSD	*.CSD, *.CICS o *.TXT

Types of files with specific extensions must necessarily be uploaded to the specified folders.

For Java J2EE Projects

An example of a structure is shown below:

- **SRC**
  - *WEBRoot*
    - *css*
    - *html*
    - *images*
    - *js*
    - *jsp*
    - *META-INF* **** Note #1, see below***
    - *WEB-INF* **** Note #2, see below***
      - *src*
      - *classes*
      - *xml*
      - *xsl*
  - *OtherJavaFiles*
- **Internal_LIB**
  - *Specific Trees for J2EE*
- **ThirdParts_LIB**
  - *Specific Trees for J2EE*

Notes:

- Note #1. This folder should also contain: wsdl / wsdd file - mapping files
- Note #2. This folder should also contain: web.xml - struts-config.xml - tld files
- The tree described will have to follow, the same structure from the deployment of the .war file, with the addition of the .java sources contained in the WEB-INF / src Folder.
- In Internal_LIB the .jar developed for the application must be inserted.
- In ThirdParts_LIB all the .jar not included in the Internal_LIB must be inserted, such as Hibernate; iBATIS; Spring; Log4j; etc. If the libraries are in an external repo and not with the folder structure or the project is not in maven, we need to copy the 3rd party jars to the source code and the java disoverer will add them to classpath.
- In the AIP Console UI you can also upload the jars and add them to classpath in the configuration part after the delivery.

For Tibco BW Projects

TIBCO Projects must respect a rigid tree, after SRC, because all the files must be archived and not only those that typically concern the Process definitions. Therefore, the structure to be adopted in the case of Business Work projects (release 5.xx - for releases above this section must be updated)

- **SRC**
  - *AESchema*
  - *Process*
  - *Connections*
  - *GlobalVar*
  - *WebServiceDEF*
  - *JavaSchema*

All the project files with the following extensions must be loaded in the specified folders. (in the case of BW 5.xx - for higher releases this section must be updated)

Type	Folder	File extension
Schema Tibco Project	AESchema	*.aeschema
Processes	Process	*.process
Connections	Connections	*.sharedjdbc *.sharedhttp *.sharedjmscon *.sharedjmsapp
Global Variables	GlobalVar	*.var; o other extensions
Java Schemas	JavaSchema	*.javaschema
WebService define	WebServiceDEF	*.wsdl

Note:

It is mandatory to insert (always in the case of BW 5.xx - for higher releases this section must be updated) also all the components that are called in the Java code (typically in the Java class methods) for example (JMS Queue Receiver for the start Processes: trigger receives for specific JMS messages, even those process starters that use SOAP, HTTP, SMTP and TCP / IP protocols

For Shell Projects

Shell projects are often used in association with other technologies in the project for asynchronous/synchronous network checks or also for example for the purposes of commercial checks on the provisioning and Billing systems. They are batch programs that are called by many shell managers such as:

- Korn shell
- Bourne shell
- C shell

The structure that should be defined is the one indicated below:

- **SRC**
  - bash
  - bsh
  - csh
  - includefiles
  - ksh
  - sh
  - shell
  - ssh
  - tsch

All the project files with the following extensions must be loaded in the specified folders.

Type	Folder	Files extensions
Bourne-Again Shell	bash	*.bash
Tenex C SHell	tsch	*.tsch
Korn Shell	ksh	*.ksh
Bourne Shell	sh	*.sh
Secure Shell	ssh	*.ssh
C Shell	csh	*.csh
Bean Shell	bsh	*.bsh
Generic Shell	shell	*.shell
Support files	includeFiles	Other extensions

Note:

It is mandatory to include in the IncludeFiles folder all the files supporting the script files.

For PHP Projects

For projects of this type of project it is necessary to archive all project files according to the trees defined below:

- **SRC**
  - *PHP*
  - *HTML*
  - *JS*
  - *INC*
  - *XML-XSL*
- **Internal_LIB**
  - *Specific Trees for PHP*
- **ThirdParts_LIB**
  - *Specific Trees for PHP*

For Dot Net Projects (Vbnet; C #)

For projects of this type, it is necessary to archive all project files according to the trees defined below. The particular type of project necessarily imposes the loading of the frameworks used by the application and all the internal and external libraries used. It is necessary to insert all the dotNet Solution both Legacy versions and last releases:

- **SRC**

        - *UI*
        - *BL*
        - *DL*
        - *INC*
        - *XML-XSL*
        - *WSDL*
        - *WCF*
        - *SilverLight*
- **Internal_LIB**
    - *Specific Trees for DotNet*
- **ThirdParts_LIB**
    - *Specific Trees for DotNet*

Note:

In the Internal_LIB folder, all the frameworks used by the application must be inserted (for example Dot Net Framework, Entity Framework, etc.) While in ThirdParts_LIB it is necessary to load all the libraries external to the application, even those of which it has been Deployed in an operating environment.

For SAP-ABAP Projects

For SAP projects it is important to respect the trees defined in the SAP repository with respect to the available version. The minimal tree that needs to be created is a single folder. The extractor will extract the code of each package in a separate folder. All folders will be under the same root folder. The analyzer will then be able to recreate the tree according to the information in the extracted code.

- **SRC**
    - *SIngle folder*
- **Internal_LIB**
    - *Specific Trees for SAP*
- **ThirdParts_LIB**
    - *Specific Trees for SAP*