ACE2024

STUDENT TRAINING GUIDE

Aras DevOps Introduction

aras

Revision MARCH 2024

Aras DevOps Introduction

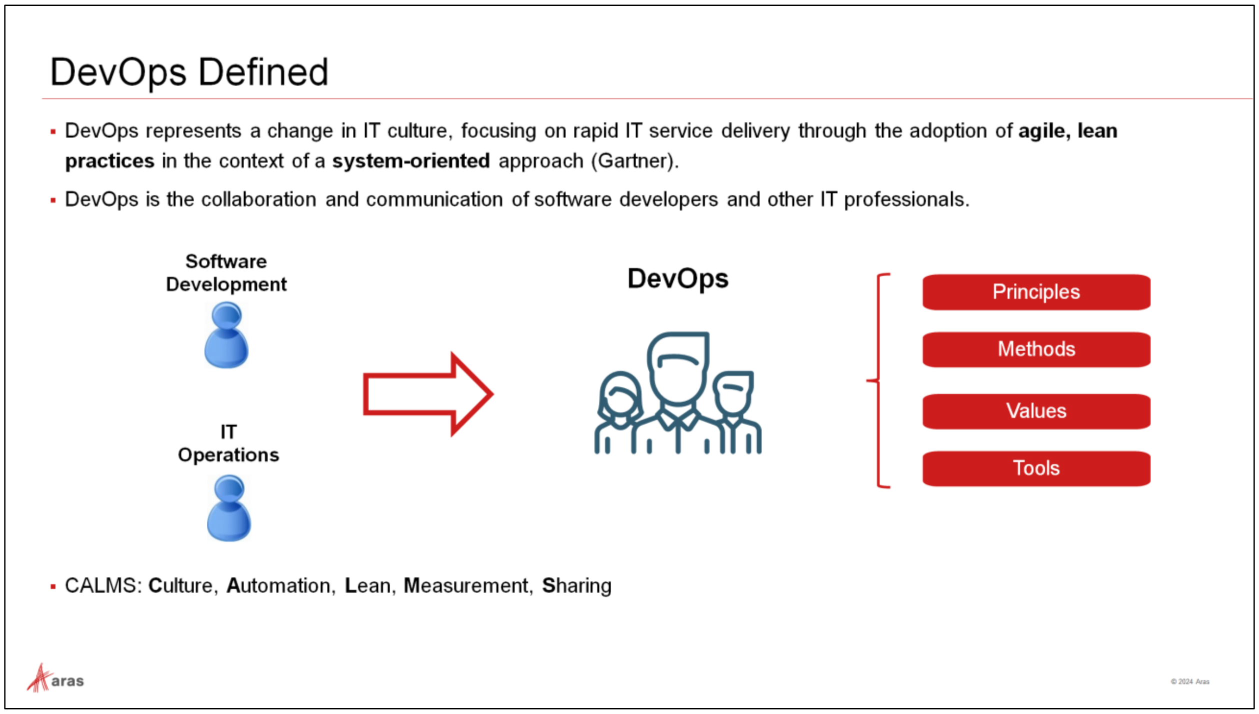
Overview

In this session, you will learn the basics, purpose, and benefits of Aras DevOps. We will also discuss the Continuous Integration and Continuous Delivery (CI/CD) processes at the heart of Aras DevOps, and the various tools used during a development project.

You will also experiment with the Aras DevOps toolchain used in the Local Development Environment (LDE) through a simple configuration use case in which you will make changes to out-of-the-box forms, export changes into the customer repository, commit the changes using Git (Source Control Management), and finally rebuild your local Innovator instance with the changed items with the **BuildAndDeploy** pipeline.

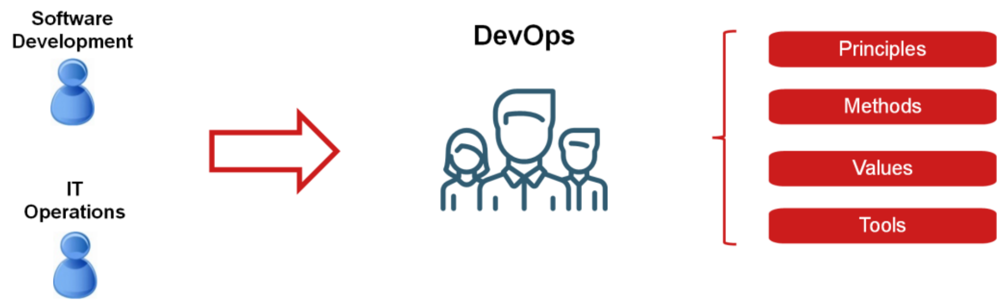
Objectives

- Understand Aras DevOps
- Understand the Standard Development Environment (SDE)
- Navigate the Local Development Environment (LDE)
- Review the components of Aras Azure DevOps
- Review the Aras DevOps Continuous Integration/Continuous Delivery (CI/CD) processes
- Understand the Aras DevOps project Git Repository structure
- State the tools included in Aras DevOps and their use
- Understand packaging in Aras DevOps
- Differentiate between packaged changes and instance specific changes
- Export changes for Aras DevOps CI/CD control
- Understand the impact of changes in working directory vs staged or committed changes



DevOps Defined

- DevOps represents a change in IT culture, focusing on rapid IT service delivery through the adoption of **agile, lean practices** in the context of a **system-oriented** approach (Gartner).
- DevOps is the collaboration and communication of software developers and other IT professionals.



- CALMS: **C**ulture, **A**utomation, **L**ean, **M**easurement, **S**haring

DevOps Defined

Culture

DevOps is a practice that emphasizes the collaboration and communication of software developers and IT Operations. But it's more than that. It's a culture change. It's tearing down silos and breaking down walls to bring groups of people together to form a team with a common set of principles, methods, values, and tools.

Automation

The process of application delivery and infrastructure management is automated. This is so that building, testing, and releasing software, as well as the deployment, configuration, and management of the infrastructure in which that software runs can happen rapidly, frequently, and more reliably. The team is accomplishing basic tasks planning, coding, building, testing, releasing, deploying, and configuring.

Lean

DevOps combines development and operations to increase the efficiency, speed, and security of software development and delivery compared to traditional processes. A nimbler software development lifecycle results in a competitive advantage for businesses and their customers. High throughput, with frequent and rapid deployment of small batch size.

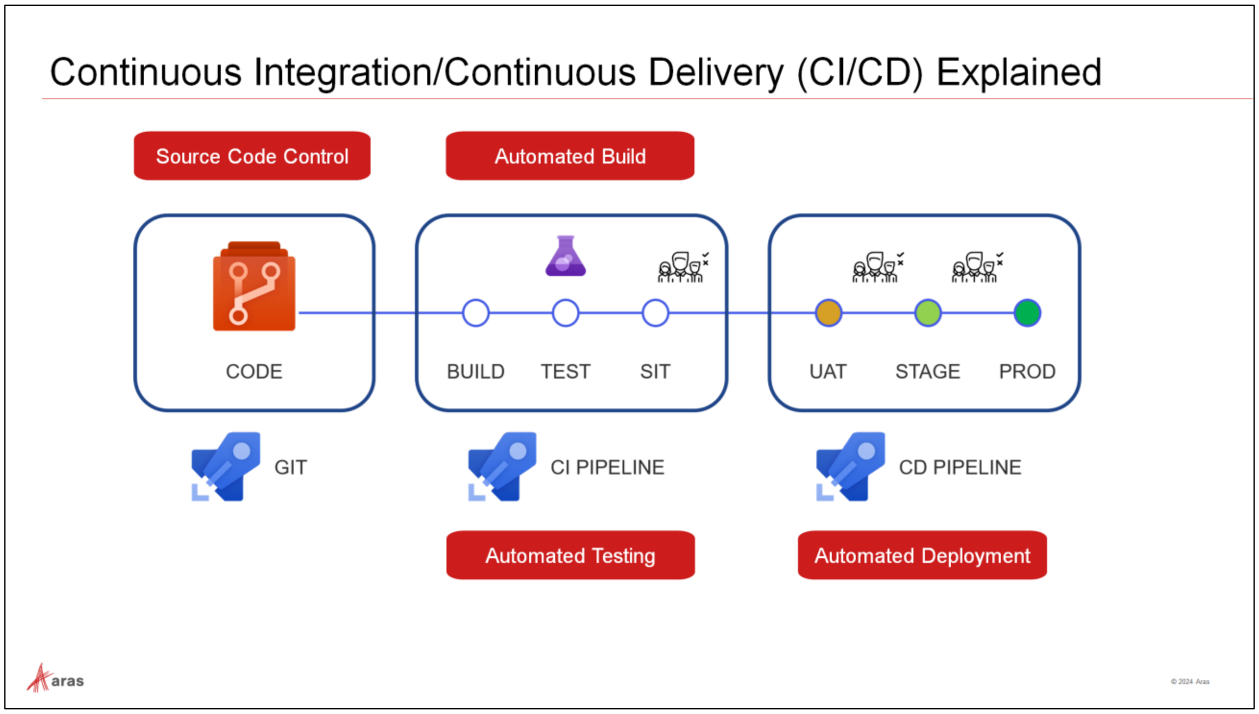
Measurements

It is important to prove continuous improvement efforts are improving the software development work. Fortunately, there are many tools and technologies for measuring performance, and assessing the proper metrics to use in your software development cycles.

Sharing

DevOps contributors work in a continuous loop, a cycle of working class functionally. Together, many different members from technical teams, business teams, the coding team, production and deployment team, and the QA team all work through the entire process to understand and meet the needs of the customer with seamless IT services. This sharing increases trust, faster software releases, more reliable deployments, and a better feedback loop between teams and customers; ultimately this helps break down the Development and Operations siloes and benefit both customers and DevOps personnel.

CALMS: **C**ulture, **A**utomation, **L**ean, **M**easurement, **S**haring



Continuous Integration/Continuous Delivery (CI/CD) Explained

Continuous Integration/Continuous Delivery (CI/CD) Explained

DevOps and CI/CD

DevOps and CI/CD tend to get used interchangeably

- DevOps is a Culture – and it is based on Continuous Integration and Continuous Delivery practices.
- You need to build and adopt processes to hit the DevOps culture.
- It starts with Code – Source code control. That code needs to be built into deliveries and artifacts that can be deployed. These deployments need to be tested and the more automated testing you have, the better the quality of the testing. Lastly deploy the packaged code into the UAT, Staging and production environments. Tying these together is the basics of the CI/CD pipeline.
- With a CI/CD pipeline, developers can make changes to code that are then automatically tested and pushed out for delivery and deployment.

CI/CD enables the delivery team to:

- Make changes on their local machines
- Commit their changes into a central repository
- Merge their changes with a pull request

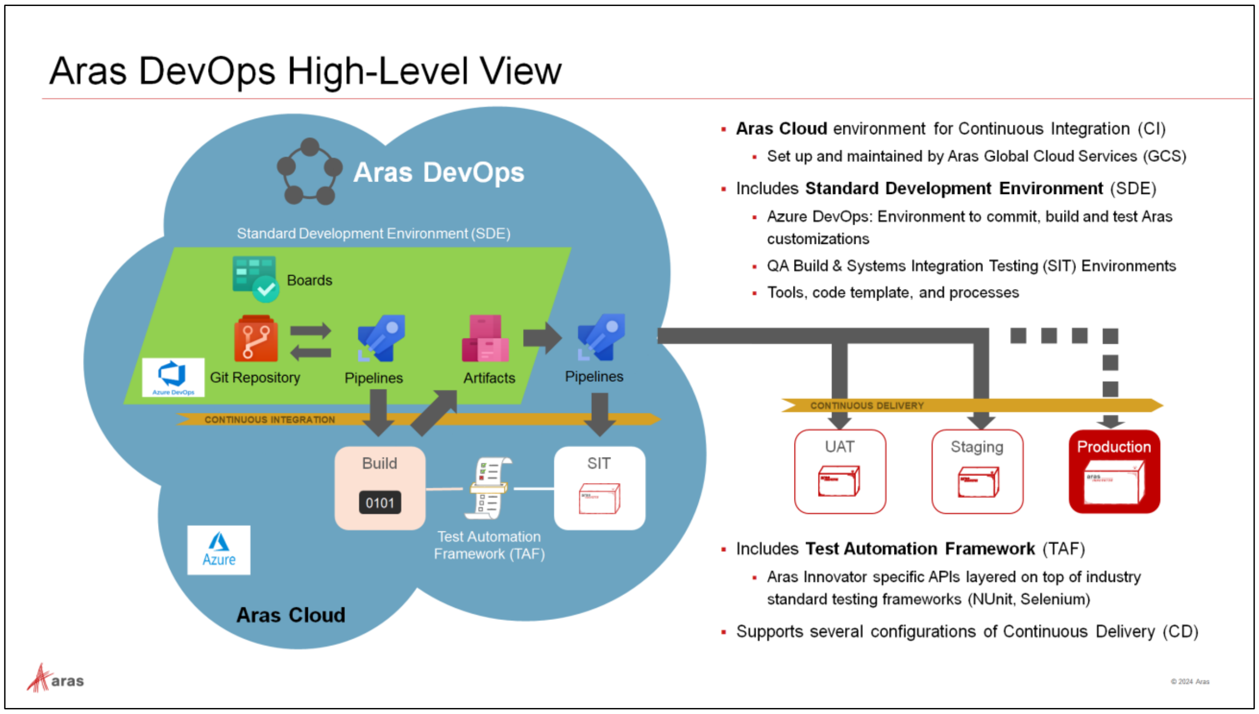
The illustration above shows the basic CI/CD flow:

- Code = source code control. Source code control includes items such as configuration files and settings. It includes the code tree and the various libraries that constitute the solution.
- Commits are means to manage changes that take solution from one configuration to the next.

- The CI Pipeline supports the continuous integration where various contributors use pull requests (PRs) to submit their contributions to the integrated whole. The system automatically builds and runs available automated tests. Reviewers check the work before accepting it. The resulting artifacts are eventually deployed to SIT for automated testing.
- The CD Pipeline supports the need to deploy the validated (SIT tested) artifacts for user acceptance test and in some cases, it is used as a basis to verify the ability to migrate existing customer data into a staging area. Issues may arise during UAT. After remediation of the issues, the solution is eventually deployed into production.

Notes:

- CODE: Changes, including code, configuration, etc.
- Source Code Control Tool: Git
- CI/CD Pipelines: For example, Jenkins, Azure Pipelines, Bamboo, etc.
- SIT: System Integration Testing
- UAT: User Acceptance Testing



Aras DevOps High-Level View

- **Aras Cloud** environment for Continuous Integration (CI)
  - Set up and maintained by Aras Global Cloud Services (GCS)
- Includes **Standard Development Environment** (SDE)
  - Azure DevOps: Environment to commit, build and test Aras customizations
  - QA Build & Systems Integration Testing (SIT) Environments
  - Tools, code template, and processes
- Includes **Test Automation Framework** (TAF)
  - Aras Innovator specific APIs layered on top of industry standard testing frameworks (NUnit, Selenium)
- Supports several configurations of Continuous Delivery (CD)

Aras DevOps High-Level View

Pre-configured Continuous Integration template

- Provides GIT for Source Control Management leveraging a Pull Request (PR) process.
- Provides Azure Pipelines for executing automated tests, code scans, etc.
- Provides Artifactory for storing all build artifacts to allow for reuse in deploying to multiple environments.
- Provides an SIT server to allow for end user/QA validation and review of work.
- Provides TAF license so your teams can build robust automated tests.
- Dedicated team to help ensure the pipeline is operational.

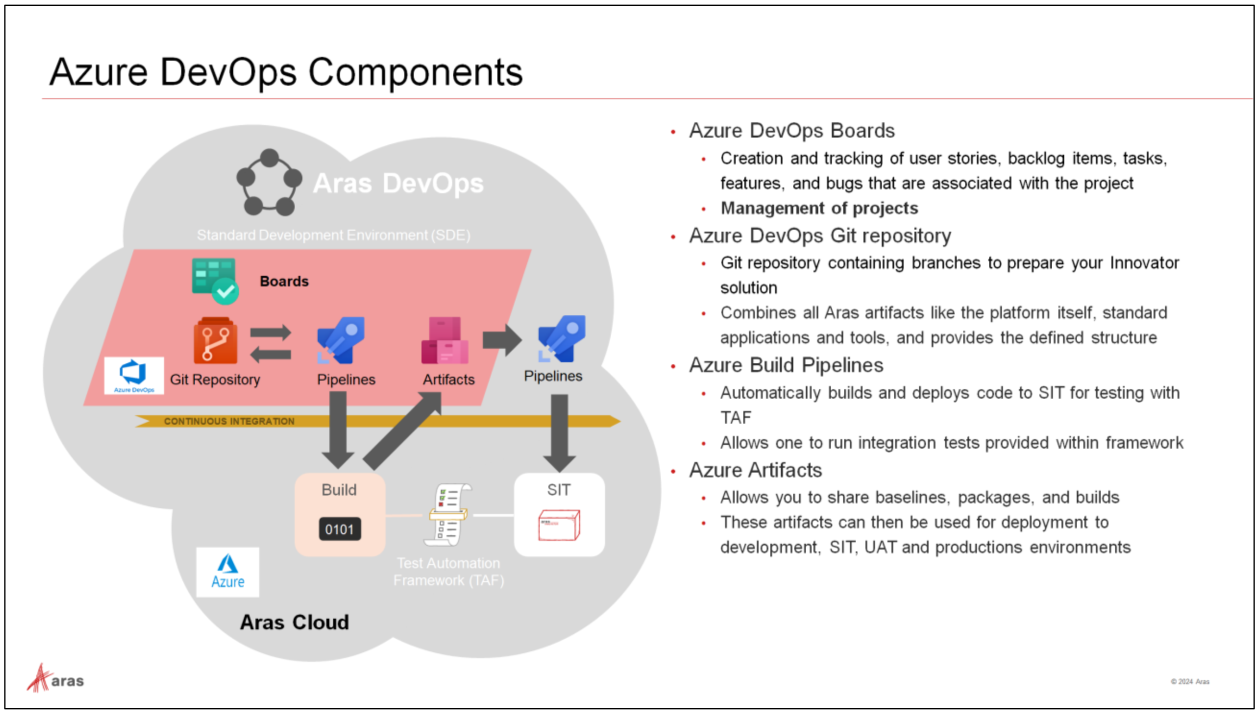
Continuous Delivery

- Aras DevOps provides the full CI experience. Continuous Delivery is reviewed on a project-by-project basis to meet customer needs.
- Aras Enterprise offering provides the full CI/CD offering for managing deployments as well as continuous integration.
- Full Continuous Delivery pipeline available instantly to support your DevOps Journey
- Continue to benefit as Aras evolves and introduces new enhancements into the Aras DevOps offerings.
- No additional internal hardware/maintenance required for supporting the pipeline.
- Uses the same tools and process leveraged by Aras Solution Delivery.
- Can be leveraged by downstream teams (e.g., support and upgrades) to help improve overall customer satisfaction.
- Includes TAF license for helping build/extend your test automation.

Standard Development Environment (SDE)

- An environment with tools and processes that enables you to adopt industry-common CI/CD practices.
- Standard Tools and Software – all tools and software (required and optional) and its integrated configuration that we are using locally for our goals (Git, Visual Studio, MS SQL and so on)
- Aras Tools – all proprietary software and solutions that we are using (Aras VS Plugin, Import/Export, TAF, …)
- Services – hosted services and applications accessible via network/Internet (Azure DevOps – base tool for the SDE CI/CD and other basic concepts)
- Industry Practices – how we are using everything listed above (Code Guidelines, Industry Best Practices: CI/CD, DevOps)

Aras Pipelines – formal definition of start to end and iterative processes with roles, activities and other detentions that helps to reach local and global goals.



Azure DevOps Components

- Azure DevOps Boards
  - Creation and tracking of user stories, backlog items, tasks, features, and bugs that are associated with the project
  - **Management of projects**
- Azure DevOps Git repository
  - Git repository containing branches to prepare your Innovator solution
  - Combines all Aras artifacts like the platform itself, standard applications and tools, and provides the defined structure
- Azure Build Pipelines
  - Automatically builds and deploys code to SIT for testing with TAF
  - Allows one to run integration tests provided within framework
- Azure Artifacts
  - Allows you to share baselines, packages, and builds
  - These artifacts can then be used for deployment to development, SIT, UAT and productions environments

Azure DevOps Components

Azure Services include Azure DevOps Git, Azure Boards, Azure DevOps Pipelines, and Azure DevOps Artifacts.

Azure Aras DevOps comes preconfigured with limited scope for changes to maintain the “standard.” In general, add-ons from the marketplace are not supported.

Azure Boards

Azure Boards can be used to plan, track, and discuss work across teams using the Agile planning tools that are available. Using Azure Boards, teams can manage their software projects. It also offers a unique set of capabilities, including native support for Scrum and Kanban. You can also create customizable dashboards, and it offers integrated reporting and integration with Microsoft Teams and Slack.

You can create and track user stories, backlog items, tasks, features, and bugs that are associated with the project using Azure Boards.

Azure Repos

Azure Repos provides support for private Git repository hosting. It offers a set of version control tools that can be used to manage the source code of every development project, large or small. When you edit the code, you ask the source control system to create a snapshot of the files. This snapshot is saved permanently so that it can be recalled later if needed.

Azure Repos offers standard Git so that developers can use the tools and clients of their choice, such as Git for Windows, Mac, third-party Git services, and tools such as Visual Studio and Visual Studio Code.

Azure Pipelines

You can use Azure Pipelines to automatically build, test, and deploy code to make it available to other users and deploy it to different targets, such as a **development, test, acceptance, and production** (**DTAP**) environment. It combines CI/CD to automatically build and deploy your code.

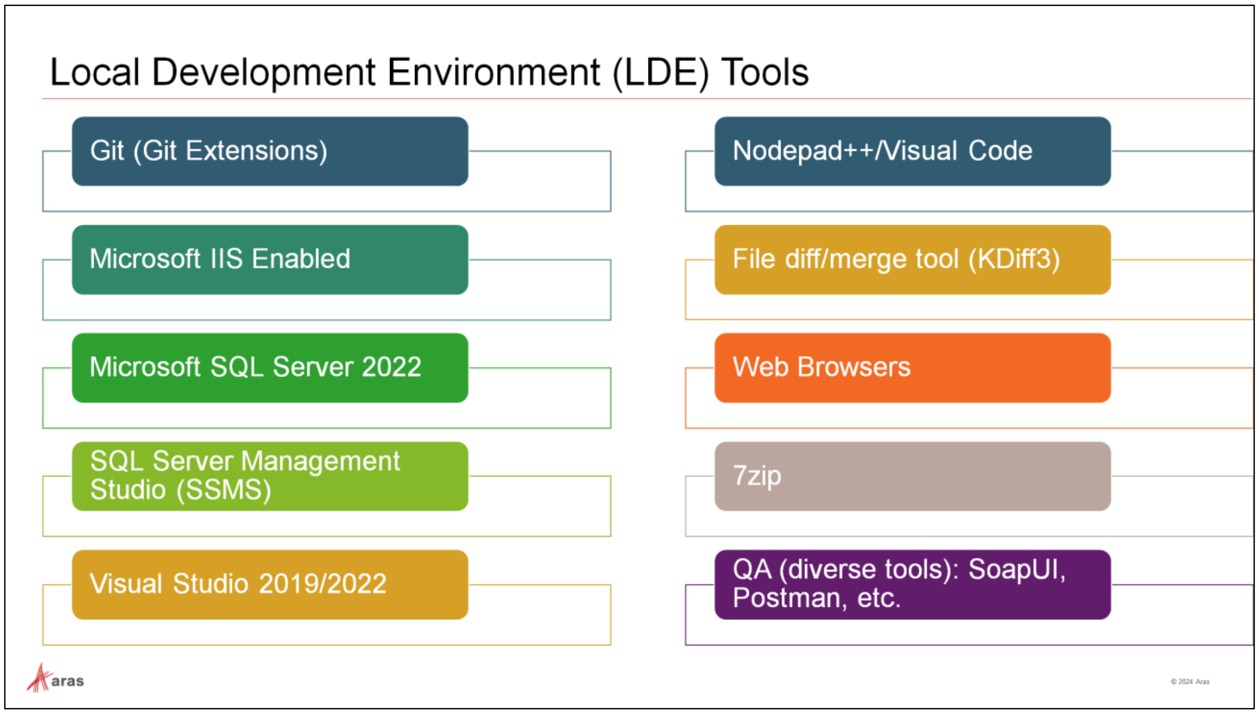
Azure Artifacts

With Azure Artifacts, you can create and share NuGet, npm, Python, and Maven packages from private and public sources with teams in Azure DevOps. These packages can be used in source code and can be made available to the CI/CD pipelines. With Azure Artifacts, you can create multiple feeds that you can use to organize and control access to the packages.

Azure Test Plans

Test Plans provide a browser-based test management solution to test your applications, including manual/exploratory testing and continuous testing. It allows users to manage test plans, test suites, and test cases for everyone in the software development process.

With Test Plans, you can ensure your Quality Assurance team uses the same Azure DevOps project as your developers, as the single source of truth, to build out their Test Plans and gather feedback from stakeholders.



Local Development Environment (LDE) Tools

- Git (Git Extensions)
- Microsoft IIS Enabled
- Microsoft SQL Server 2022
- SQL Server Management Studio (SSMS)
- Visual Studio 2019/2022
- Nodepad++/Visual Code
- File diff/merge tool (KDiff3)
- Web Browsers
- 7zip
- QA (diverse tools): SoapUI, Postman, etc.

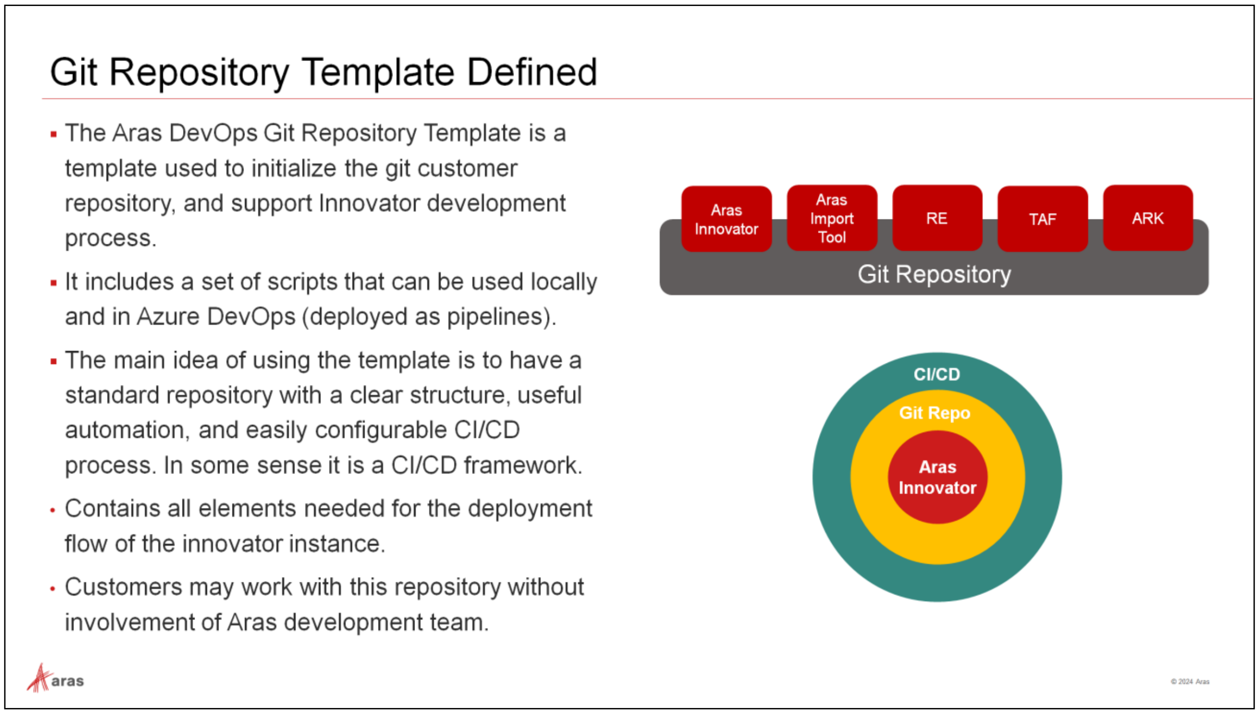
aras

© 2024 Aras

Local Development Environment (LDE) Tools

Required and Optional Tools

- Git (use latest version)
- Git Extensions (not required but useful for learning/training purposes)
- Microsoft IIS Enabled (according to Innovator Installation Guide)
- MSSQL Server 2019 or 2022 (according to Innovator Installation Guide)
- SSMS SQL Server Management Studio (to manage SQL server useful for learning/training purposes)
- Visual Studio Community Edition (or Pro) 2019 or 2022, if tests will be developed
  - Aras Method Plugin
  - ReSharper Standard Edition at least (in case of heavy performance work – needs Ultimate edition) – VS extension to help write code faster and to debug performance issues
- File diff/merge tool (KDiff3)
- Web Browsers
- 7zip (installed by Aras DevOps)
- Visual Code/Notepad++, or other preferred text editor (not required but useful for learning/training purposes)
- QA tools
  - Greenshot (for free) or similar; Fiddler (for free); Recordit (for free) or similar; SoapUI/Postman
  - Test Management System
  - Total Commander (any version) – for comparing files and folders



Git Repository Template Defined

- The Aras DevOps Git Repository Template is a template used to initialize the git customer repository, and support Innovator development process.
- It includes a set of scripts that can be used locally and in Azure DevOps (deployed as pipelines).
- The main idea of using the template is to have a standard repository with a clear structure, useful automation, and easily configurable CI/CD process. In some sense it is a CI/CD framework.
- Contains all elements needed for the deployment flow of the innovator instance.
- Customers may work with this repository without involvement of Aras development team.

aras

© 2024 Aras

Git Repository Template Defined

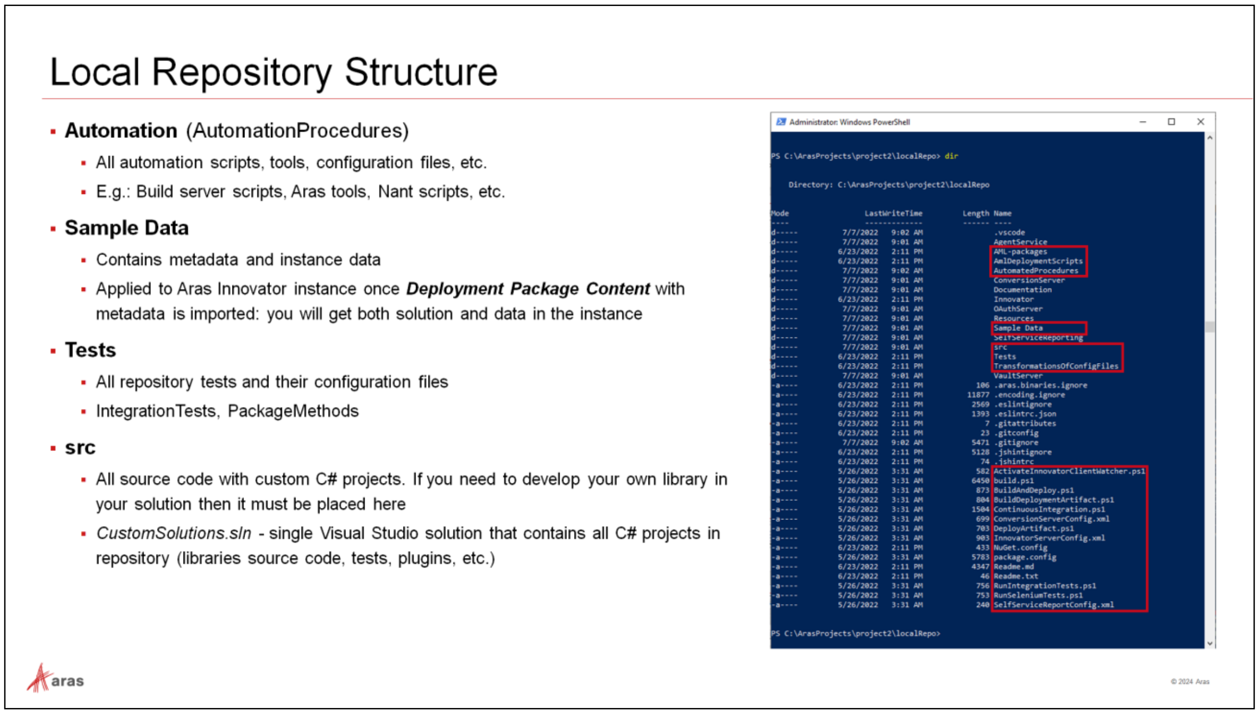
The Aras DevOps Git Repository Template is a template used to initialize the git customer repository, and support Innovator development process.

The main idea of using the template is to have a standard repository with a clear structure, useful automation, and an easily configurable CI/CD process. In some sense it is a CI/CD framework. It combines all Aras artifacts like the platform itself, standard applications and tools and provides the defined structure.

From the picture you should understand what the Git Repository consists of. It is the entire Aras Innovator platform in a ***Baseline*** format, but instead of storing the *DB.bak* that can be huge, we export all ***AML packages* metadata** via ***Aras Export Tool*** and commit them. The crucial word here is **metadata** that means only Items definitions without thousands of instantiated objects. As you know, the ***Baseline*** does not separate logical blocks, and therefore Aras applications such as RE (Requirements Engineering) and TAF (Test Automation Framework) blocks from the schema will be in the ***Git Repository*** as a ***Baseline*** part. ***Aras Import Tool*** as well as ***Aras Export Tool***, ***ConsoleUpgrade*** and ***LanguageTool*** are stored in the ***Git Repository*** as they are.

Notes:

- RE: Requirements Engineering
- TAF: Test Automation Framework
- ARK: Aras Reusable Kit (a common name for the internal reusable Aras solutions)



Local Repository Structure

- **Automation** (AutomationProcedures)
  - All automation scripts, tools, configuration files, etc.
  - E.g.: Build server scripts, Aras tools, Nant scripts, etc.
- **Sample Data**
  - Contains metadata and instance data
  - Applied to Aras Innovator instance once ***Deployment Package Content*** with metadata is imported: you will get both solution and data in the instance
- **Tests**
  - All repository tests and their configuration files
  - IntegrationTests, PackageMethods
- **src**
  - All source code with custom C# projects. If you need to develop your own library in your solution then it must be placed here
  - *CustomSolutions.sln* - single Visual Studio solution that contains all C# projects in repository (libraries source code, tests, plugins, etc.)

```
PS C:\ArasProjects\project2\localRepo> dir


    Directory: C:\ArasProjects\project2\localRepo


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          7/7/2022   9:02 AM                .vscode
d-----          7/7/2022   9:01 AM                AgentService
d-----         6/23/2022   2:11 PM                AML-packages
d-----         6/23/2022   2:11 PM                AmlDeploymentScripts
d-----          7/7/2022   9:02 AM                AutomatedProcedures
d-----          7/7/2022   9:01 AM                ConversionServer
d-----          7/7/2022   9:01 AM                Documentation
d-----         6/23/2022   2:11 PM                Innovator
d-----          7/7/2022   9:01 AM                OAuthServer
d-----          7/7/2022   9:01 AM                Resources
d-----          7/7/2022   9:01 AM                Sample Data
d-----          7/7/2022   9:01 AM                SelfServiceReporting
d-----          7/7/2022   9:01 AM                src
d-----         6/23/2022   2:11 PM                Tests
d-----         6/23/2022   2:11 PM                TransformationsOfConfigFiles
d-----          7/7/2022   9:01 AM                VaultServer
-a----         6/23/2022   2:11 PM            106 .aras.binaries.ignore
-a----         6/23/2022   2:11 PM          11877 .encoding.ignore
-a----         6/23/2022   2:11 PM           2569 .eslintignore
-a----         6/23/2022   2:11 PM           1393 .eslintrc.json
-a----         6/23/2022   2:11 PM              7 .gitattributes
-a----         6/23/2022   2:11 PM             23 .gitconfig
-a----          7/7/2022   9:02 AM           5471 .gitignore
-a----         6/23/2022   2:11 PM           5128 .jshintignore
-a----         6/23/2022   2:11 PM             74 .jshintrc
-a----         5/26/2022   3:31 AM            582 ActivateInnovatorClientWatcher.ps1
-a----         5/26/2022   3:31 AM           6450 build.ps1
-a----         5/26/2022   3:31 AM            873 BuildAndDeploy.ps1
-a----         5/26/2022   3:31 AM            804 BuildDeploymentArtifact.ps1
-a----         5/26/2022   3:31 AM           1504 ContinuousIntegration.ps1
-a----         5/26/2022   3:31 AM            699 ConversionServerConfig.xml
-a----         5/26/2022   3:31 AM            703 DeployArtifact.ps1
-a----         5/26/2022   3:31 AM            903 InnovatorServerConfig.xml
-a----         6/23/2022   2:11 PM            433 NuGet.config
-a----         5/26/2022   3:31 AM           5783 package.config
-a----         6/23/2022   2:11 PM           4347 Readme.md
-a----         6/23/2022   2:11 PM             46 Readme.txt
-a----         5/26/2022   3:31 AM            756 RunIntegrationTests.ps1
-a----         5/26/2022   3:31 AM            753 RunSeleniumTests.ps1
-a----         5/26/2022   3:31 AM            240 SelfServiceReportConfig.xml


PS C:\ArasProjects\project2\localRepo>
```

aras

© 2024 Aras

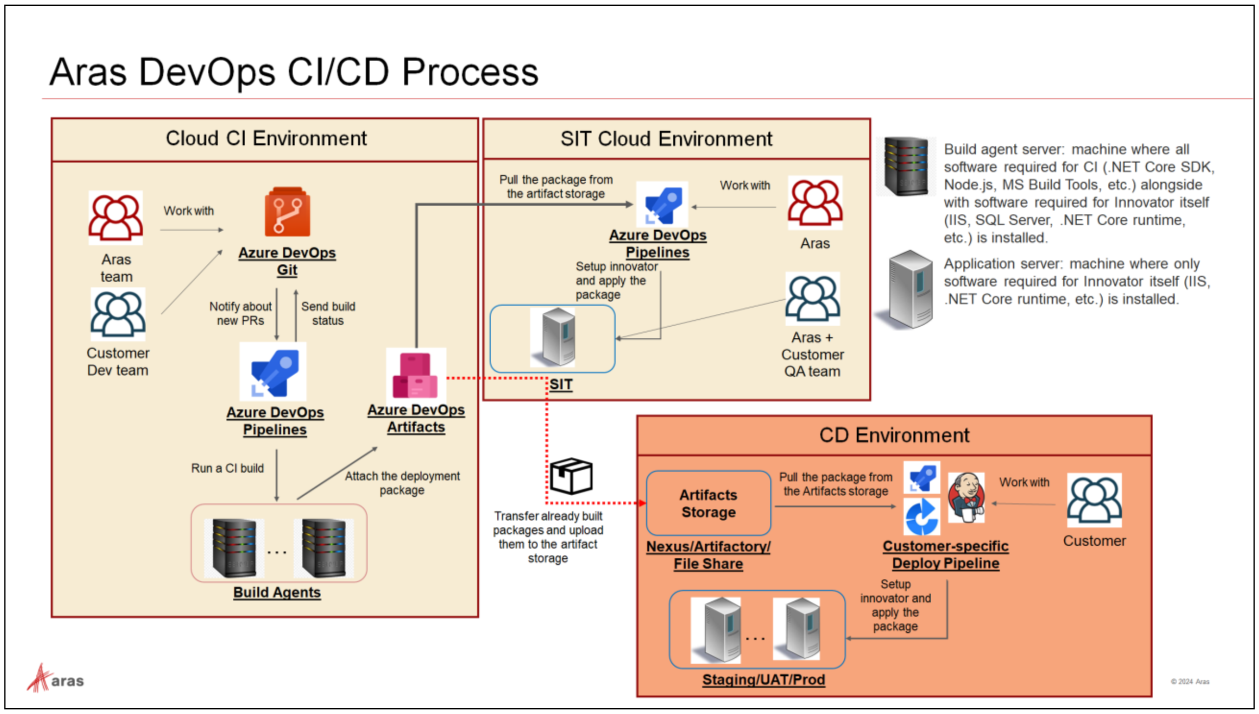
Local Repository Structure

Contents

- Automated scripts, batch files, tools required to setup CI and CD pipelines on Azure DevOps.
- Place holder for Scripts to update Innovator database.
- Place holder for Code tree files to be updated.
- Configuration files and templates used for CI and CD.
- Framework to develop, test and maintain your custom code.

Sample Data

- Sometimes it is useful to have not only a solution ***Deployment Package*** containing *metadata* (ItemTypes, Methods, Identities and so on) but also some package with *data* corresponding to this metadata (concrete Items, Users, etc.). Given the fact that *data* and *metadata* are "Items" in Aras terms, we consider the ***Sample Data*** package to be a ***Deployment Package*** (***Deployment Package Content*** to be more precise) from a structural point of view.
- ***Sample Data*** - a special form of ***Deployment Package Content*** with data like Users, Items, and so on. Unlike usual ***Deployment Package Content***, it will contain exactly data rather than solution/functionality. Such a package is being applied to Aras Innovator instance once ***Deployment Package Content*** with metadata is imported. As a result, you will get both solution and data in the instance.


Aras DevOps CI/CD Process
Cloud CI Environment
Aras team
Work with
Azure DevOps Git
Customer Dev team
Notify about new PRs
Send build status
Azure DevOps Pipelines
Azure DevOps Artifacts
Run a CI build
Attach the deployment package
Build Agents
SIT Cloud Environment
Pull the package from the artifact storage
Work with
Azure DevOps Pipelines
Aras
Setup innovator and apply the package
Aras + Customer QA team
SIT
Transfer already built packages and upload them to the artifact storage
CD Environment
Artifacts Storage
Pull the package from the Artifacts storage
Work with
Customer
Nexus/Artifactory/ File Share
Customer-specific Deploy Pipeline
Setup innovator and apply the package
Staging/UAT/Prod
aras
© 2024 Aras

Build agent server: machine where all software required for CI (.NET Core SDK, Node.js, MS Build Tools, etc.) alongside with software required for Innovator itself (IIS, SQL Server, .NET Core runtime, etc.) is installed.

Application server: machine where only software required for Innovator itself (IIS, .NET Core runtime, etc.) is installed.

Aras DevOps CI/CD Process

Cloud CI Environment

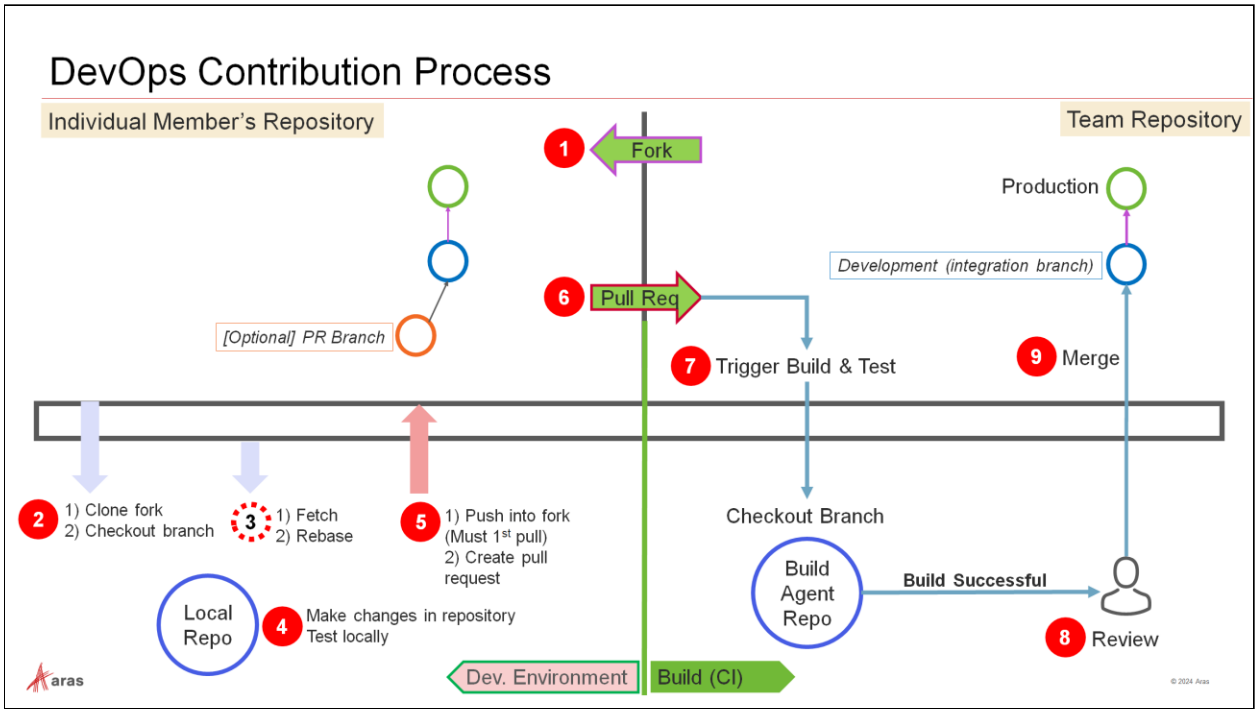
The Aras and Customer Dev teams may work together to contribute to the project's codebase. At some point their contribution will be pushed to the Azure DevOps Git Repo; a Pull Request (PR) is then created, and a CI build process is triggered to ensure the codebase passes the necessary checks. If all policy checks have passed successfully and the PR review team is satisfied with the outcome, the contribution may be merged into the central codebase. The build files are then stored in the Azure DevOps Artifacts.

Cloud SIT Environment

Deploying on SIT allows developers and testers to assess the behavior and performance of the Aras Innovator in a simulated production-like setting before it is deployed to the actual production environment.

CD Environment

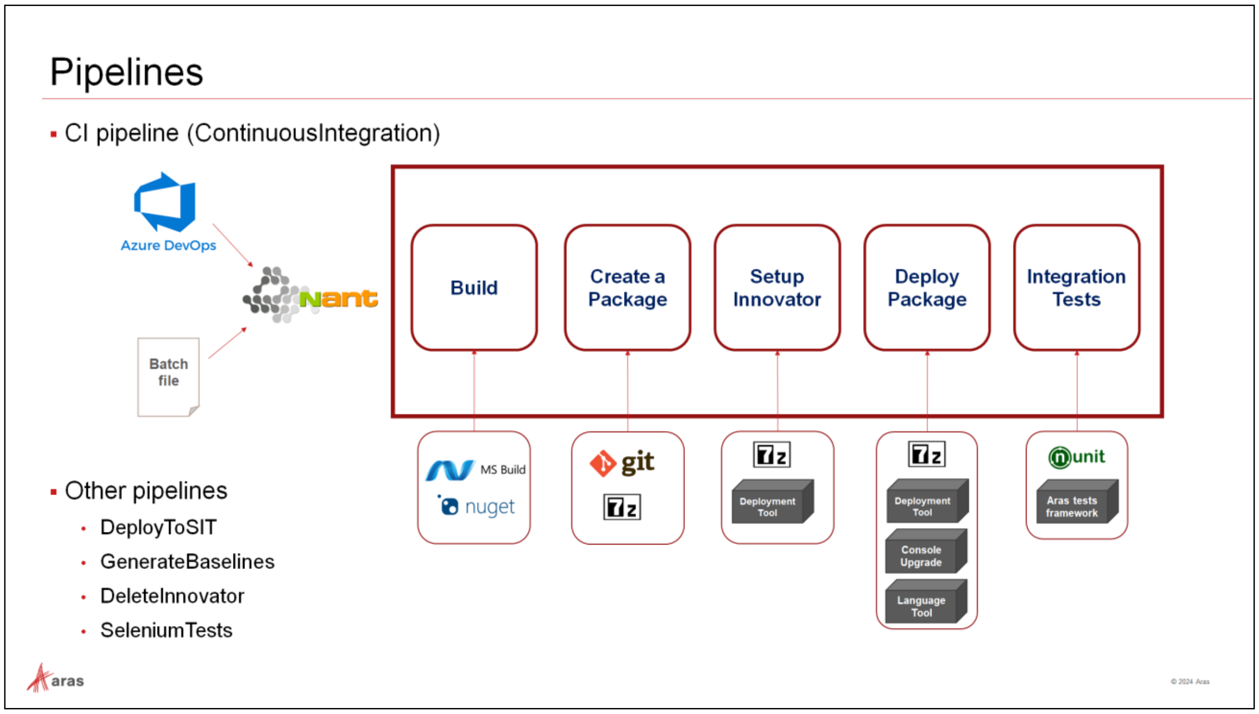
Customer, Aras's teams, may work now on deploying and validating the build package in Staging/UAT/Production environments.


DevOps Contribution Process
Individual Member's Repository
Team Repository
1 Fork
Production
Development (integration branch)
6 Pull Req
[Optional] PR Branch
7 Trigger Build & Test
9 Merge
2 1) Clone fork 2) Checkout branch
3 1) Fetch 2) Rebase
5 1) Push into fork (Must 1st pull) 2) Create pull request
Checkout Branch
Build Agent Repo
Build Successful
Local Repo
4 Make changes in repository Test locally
8 Review
Dev. Environment
Build (CI)
aras
© 2024 Aras

DevOps Contribution Process

Each developer in a team is considered a contributor, as are test/QA engineers and others. The developers and others who may contribute source files (configuration files, source code C#, etc.) need to create a Fork in which to manage their work in the SDE.

Step	Description
1	Create a Fork (copy of the shared team repository) in SDE (Aras Azure DevOps).
2	Clone your Fork to your local environment and check out the branch you wish to work on.
3	This is a step to repeat often (at least once a day before starting your work). You need to add a remote reference to the team repo so that you can fetch and rebase your current work on it. This process reduces (or even eliminates) the chance that your eventual PR (Pull Request) is rejected because of conflicts. You should also do the same with your Fork, particularly if others have access to it and can update it.
4	Do your work – execute **RunIntegrationTests**, **ContinuousIntegration**, and other scripts locally.
5	Commit your changes locally and then push to Fork.
6	Create your pull request in SDE.
7	Updates to your source branch after your pull request is created and the initial branch creation trigger a **ContinuousIntegration** pipeline execution based on branch policy.
8	When the **ContinuousIntegration** pipeline has successfully built your artifact, it will run tests and report a success status (green) or failure status (red). Based on that status, reviewers may approve your PR. They may also reject it even with a green build, if some other project practice is violated.
9	When the PR is approved and merged, the branch would typically have a policy to run the **ContinuousIntegration** pipeline again. This is to make sure that all PRs are still in sync. The goal is to ensure that the common repo branch is always buildable.



Pipelines

- CI pipeline (ContinuousIntegration)
- Other pipelines
  - DeployToSIT
  - GenerateBaselines
  - DeleteInnovator
  - SeleniumTests

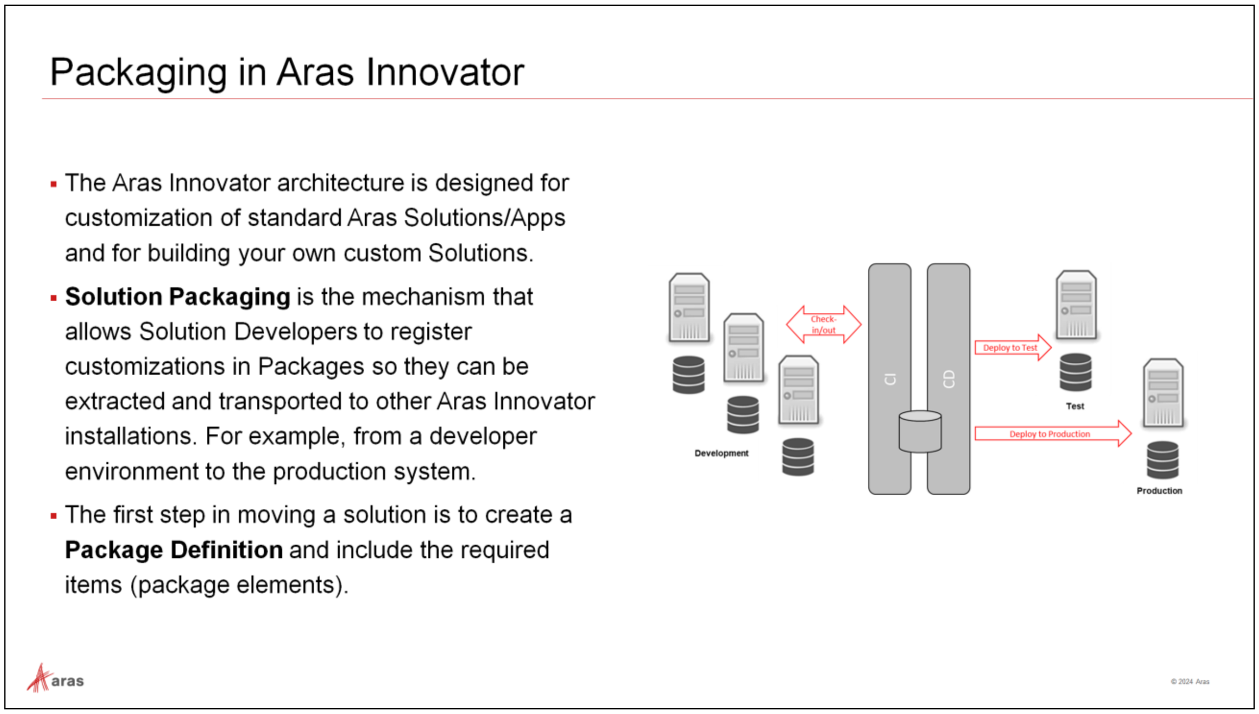
Pipelines

Azure DevOps Pipelines is the automation server used to automatically create application builds; batch files may also be used. In Aras DevOps, all these stages logic is implemented in NAnt, which, in turn, runs a lot of Aras (black boxes on the schema) and 3rd party tools.

An abstract CI/CD pipeline which is used in Aras DevOps is presented:

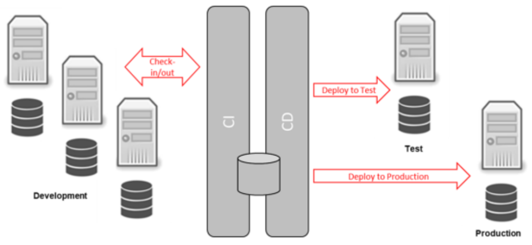
Stages

- Build
  - Build custom VS solutions
  - Build tests
- Create a package
  - Calculate files changed since last release
  - Package them
- Set up Innovator
  - Restore temporary Innovator instance from a baseline
  - Set up configs
- Deploy Package
  - Apply a package to the instance, to update since baseline to a new state
- Integration Tests
  - Run integration tests using Aras TAF



Packaging in Aras Innovator

- The Aras Innovator architecture is designed for customization of standard Aras Solutions/Apps and for building your own custom Solutions.
- **Solution Packaging** is the mechanism that allows Solution Developers to register customizations in Packages so they can be extracted and transported to other Aras Innovator installations. For example, from a developer environment to the production system.
- The first step in moving a solution is to create a **Package Definition** and include the required items (package elements).



© 2024 Aras

Packaging in Aras Innovator

Aras Innovator is a low-code platform, which means users can add truly little code to achieve outstanding results rapidly. It also means user can use configuration to express business rules, such as a life cycle map.

When working directly with an instance of Aras Innovator, these changes are stored anonymously within and can reference any other items already in the system and vice versa.

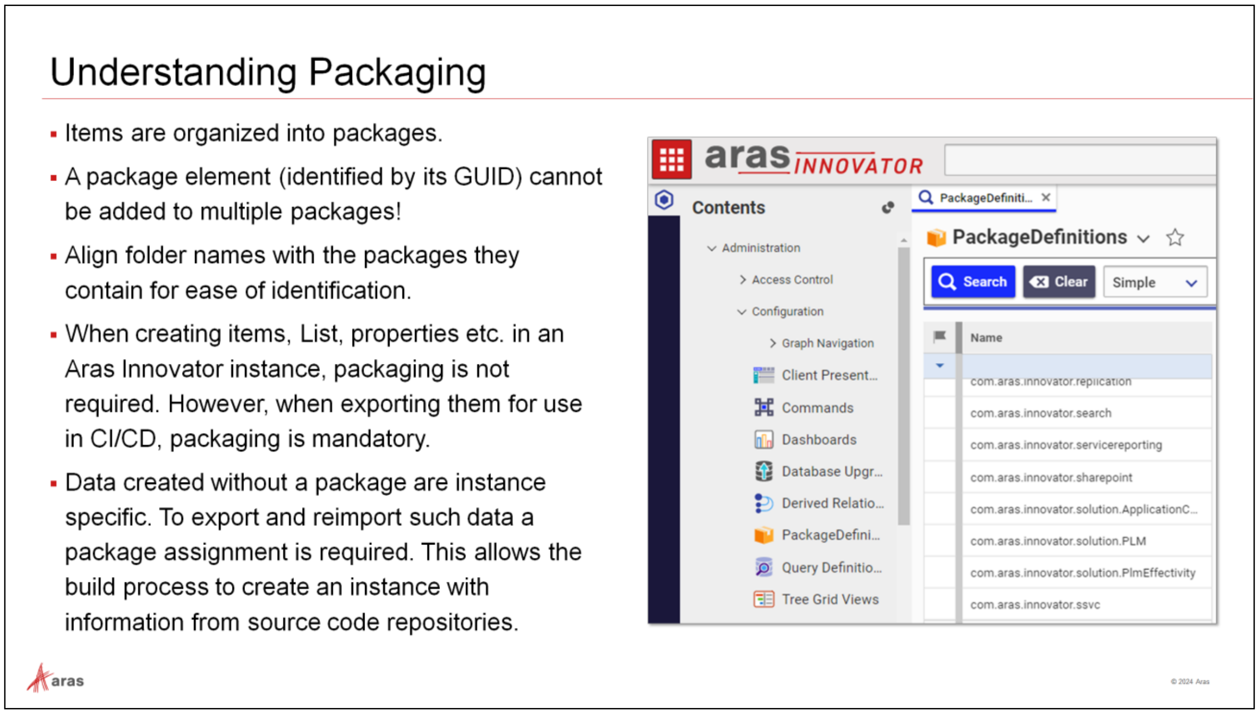
Making such changes directly in a business-critical system serving users is not a good practice. As mentioned earlier, a central focus of DevOps is to instill the discipline of well-managed solution configurations, including change management and implementation.

In larger solution development engagements, exported packages and their elements can be put under version control and can be input to an automated build process (CI/CD).

The following sections explain how to export these changes into named packages, define explicit dependencies, and commit the changes for proper configuration and version control.

This way, it is ensured that the build system can replicate the swiftly accomplished interactive tasks - thus introducing configuration and version control discipline to the low-code product.

The first step in this procedure is defining a package by a **Package Definition** item, which then can be exported and re-imported to the central source control system for further usage in the builds.



Understanding Packaging

- Items are organized into packages.
- A package element (identified by its GUID) cannot be added to multiple packages!
- Align folder names with the packages they contain for ease of identification.
- When creating items, List, properties etc. in an Aras Innovator instance, packaging is not required. However, when exporting them for use in CI/CD, packaging is mandatory.
- Data created without a package are instance specific. To export and reimport such data a package assignment is required. This allows the build process to create an instance with information from source code repositories.

Name
com.aras.innovator.replication
com.aras.innovator.search
com.aras.innovator.servicereporting
com.aras.innovator.sharepoint
com.aras.innovator.solution.ApplicationC...
com.aras.innovator.solution.PLM
com.aras.innovator.solution.PlmEffectivity
com.aras.innovator.ssvc

Understanding Packaging

Conceptually each package is a collection of item IDs. Additionally certain ItemTypes were introduced to support package functionality.

A newly installed Aras Innovator database contains Package Definitions of two types:

- **Aras Core Packages** – These packages are used to define the basic structure of every Aras Innovator database, regardless of what solutions are used in the database. They are created and managed by Aras for the Core system.
  Do not modify, because any modifications (addition, change of value, deletion) will put your Aras Innovator installation functionality at risk.
- **Aras Solution Packages** – These packages define the elements that comprise the definition and functional rules of different Solutions data models. They are created and managed by Aras for a dedicated Solution. The names typically begin with *com.aras.innovator.solution*. Modify with care!

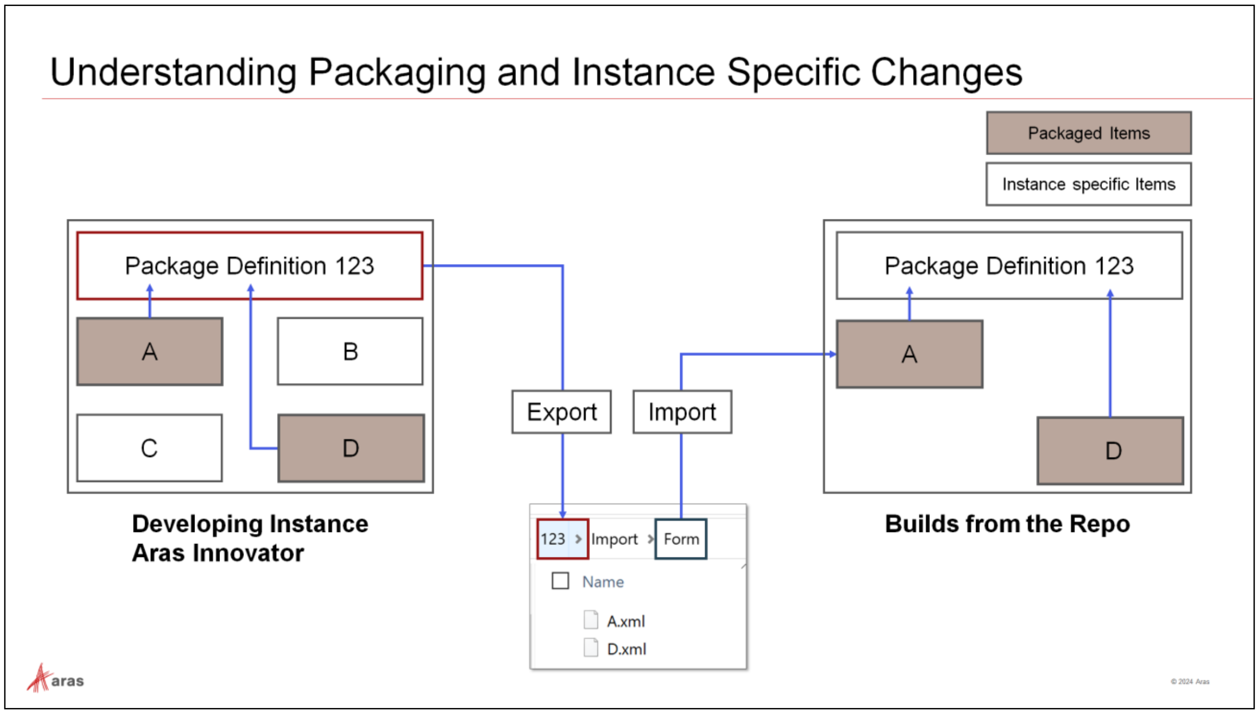
Try It … Explore existing Package Definitions

1. Navigate to **Administration** > **Configuration** > **PackageDefinitions**.
2. Run a blank search.
3. Open some of the Package Definitions, and note their structure, for example **com.aras.innovator.core** or **com.aras.innovator.solution.PLM**.

Notes:

- A package name must be unique within a database of an Aras installation, and it can be any text up to 64 characters (excluding special characters; spaces are allowed).
- Package names should be globally unique, at least within your Aras ecosystem, and should follow a naming convention. We recommend using a "." notation for names. A good example for this would be to follow the java package naming conventions.

- Standard Aras packages from Applications and Core modules already follow a convention. The namespaces defined in this convention are reserved for Aras internal packages and shall not be applied outside of Aras!



Understanding Packaging and Instance Specific Changes

Aras Innovator is very flexible in enabling users to do rapid application development and proofing out concepts.

This flexibility requires management to satisfy good practices developed in the industry to manage configuration of enterprise systems. ITIL (Information Technology Infrastructure Library) has such controls in the form of practices in the latest versions. SOC 2 (System and Organization Controls 2) certification also requires strong configuration (change) management.

For such reasons, administrators must adopt the discipline of extracting such changes and managing them in DevOps. If for no other reason than to ensure the next release does not wipe out their changes.

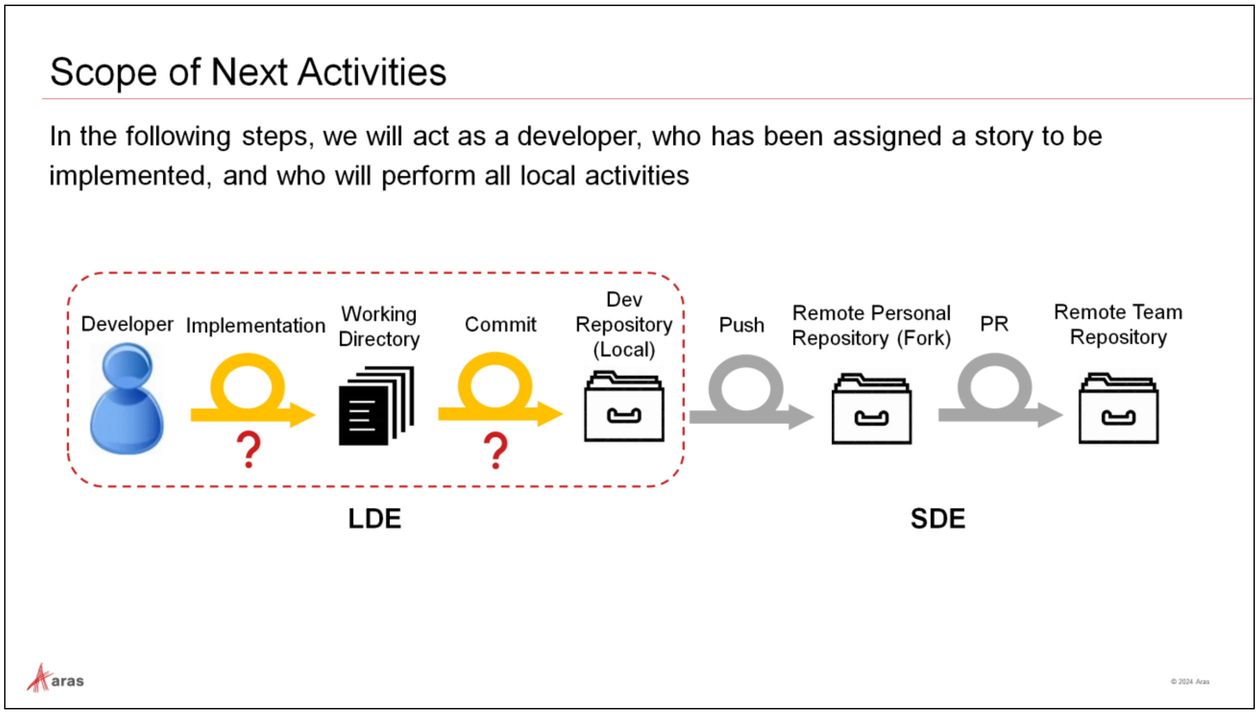
When changes are properly change-controlled then the corresponding solution configuration can be reproduced with assurance.

The diagram summarizes the impact of anonymous items in an instance and the effect of defining, exporting, and providing packages to the build system.

On the left, the user has four items (A, B, C, and D). **A**, and **D** represent new or modified items which are properly packaged, exported, copied, and assigned to the change control system Git. **B** and **C** represent Innovator instance specific items which are not intended for use in the next build and therefore consequently not packaged.

The build system then produces the instance on the right. It is important to observe that the Aras Innovator instance on the right excludes items **B** and **C**, highlighting the capability to dictate what DevOps builds. This capacity to specify what DevOps creates is a foundational element of utilizing DevOps.

Note: The visualization is reduced to changed items only.



Scope of Next Activities

In the following steps, we will act as a developer, who has been assigned a story to be implemented, and who will perform all local activities

Scope of Next Activities

In the next steps we will implement a small User Story and learn how to prepare changes to be committed. We will create commits with the use-case implementation in the local repository.

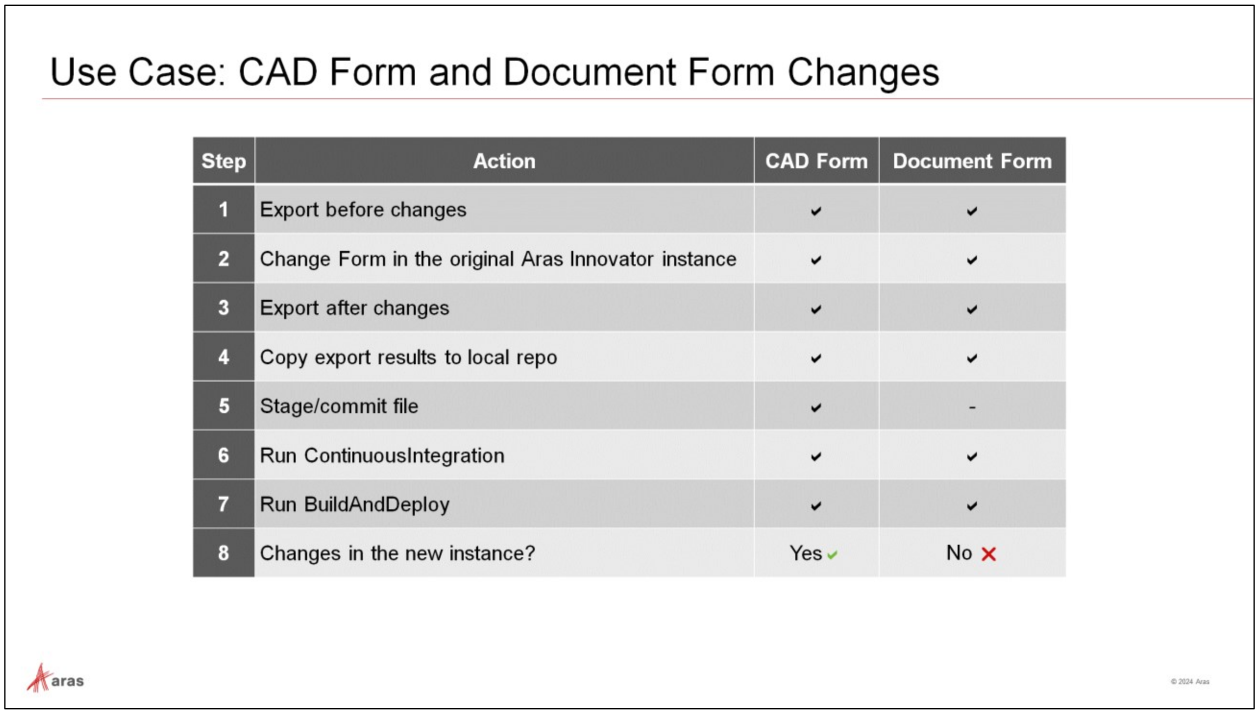
Work in LDE

- The project development work for each developer begins in their Local Development Environment (LDE), after they ensured that the **trainingxx** branch has been checked out (this branch was created in an earlier step from the Team's repository corresponding branch).
- Once each developer has completed their local work, they commit their user story contribution to their local repository and ensure that it is validated with the **ContinuousIntegration** and/or **BuildAndDeploy** pipelines.
- Each developer performs a **git fetch/rebase** operation from the appropriate Team's repository branch to ensure that they are fully synchronized. Any conflict that occurs because of this operation should be fixed before taking the next steps.

Work in SDE

- Each developer performs a **git push** operation to their personal fork's **trainingxx** branch.
- In Azure DevOps, each developer creates an appropriate Pull Request (PR) to present their work to the review team.
- If the developer's contribution is accepted it is merged into the Team's repository **trainingxx** branch.

Note: When working on one feature at a time then only the "development" branch is necessary.



Use Case: CAD Form and Document Form Changes

Step	Action	CAD Form	Document Form
1	Export before changes	✓	✓
2	Change Form in the original Aras Innovator instance	✓	✓
3	Export after changes	✓	✓
4	Copy export results to local repo	✓	✓
5	Stage/commit file	✓	-
6	Run ContinuousIntegration	✓	✓
7	Run BuildAndDeploy	✓	✓
8	Changes in the new instance?	Yes ✓	No ✗

aras

© 2024 Aras

Use Case: CAD Form and Document Form Changes

The idea behind the following use case is to make changes to two items that already exist in the baseline, and to integrate the changes to the future builds. The two items are the CAD Form and the Document Form.

For one of the two items we perform all necessary steps, and we will have success; the other one is used to demonstrate a possible error when one skips the important steps of staging and committing any changes to Git.

Since you have added the properties to existing items, there is no need to update the import manifest file in this use case, as no new package has been added to the solution: we only modified files.

The subsequent import during the build works because it is an amendment to the package that already exists in the baseline.

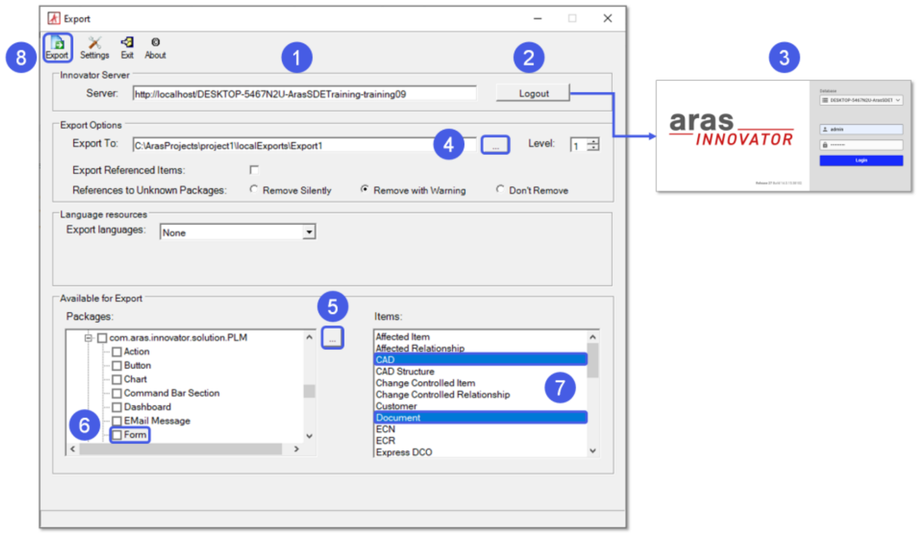
The use case steps include checks and verifications which help to understand the details of the process as listed in the table.

Step	Action	CAD Form	Document Form	Result
1	Export before changes	✓	✓	C:\ArasProjects\project\localExports\FormExport1\PLM
2	Change Form in the original Aras Innovator instance	✓	✓	Forms in instance have one more field each
3a	Export after changes	✓	✓	C:\ArasProjects\project\localExports\FormExport2\PLM
3b	View differences	✓	✓	Differences in both forms
3c	Review the manifest file in the local repo and in the export results	✓	✓	The manifest file is unchanged

Step	Action	CAD Form	Document Form	Result
4	Copy export results to local repo	✓	✓	Local repo contains the changes in the files
5a	Stage/commit the xml file in the Form folder	✓	–	Only the modified file for the CAD form is in Git
5b	Confirm change in Git	✓	✓	Git contains changes as non-staged files
6a	Run ContinuousIntegration	✓	✓	No error
6b	Confirm change in Innovator instance	✓	✓	Forms in instance still have the new field
7	Run BuildAndDeploy	✓	✓	No error
8	Confirm the changes in the new Innovator instance	Yes	No	CAD: ok Document: like before the change

Exporting CAD and Document Forms Before Changes

Step 1



Server: http://localhost/DESKTOP-5467N2U-ArasSDETraining-training09

Export To: C:\ArasProjects\project1\localExports\Export1

Level: 1

Export Referenced Items: ☐

References to Unknown Packages: Remove Silently [✓ Remove with Warning] Don't Remove

Export languages: None

Packages:

- ☐ com.aras.innovator.solution.PLM
  - ☐ Action
  - ☐ Button
  - ☐ Chart
  - ☐ Command Bar Section
  - ☐ Dashboard
  - ☐ EMail Message
  - ☐ Form

Items:

- Affected Item
- Affected Relationship
- [✓ CAD]
- CAD Structure
- Change Controlled Item
- Change Controlled Relationship
- Customer
- [✓ Document]
- ECN
- ECR
- Express DCO

aras

© 2024 Aras

Exporting CAD and Document Forms Before Changes

Export the changes to your export folder for comparison later with your exported changes.

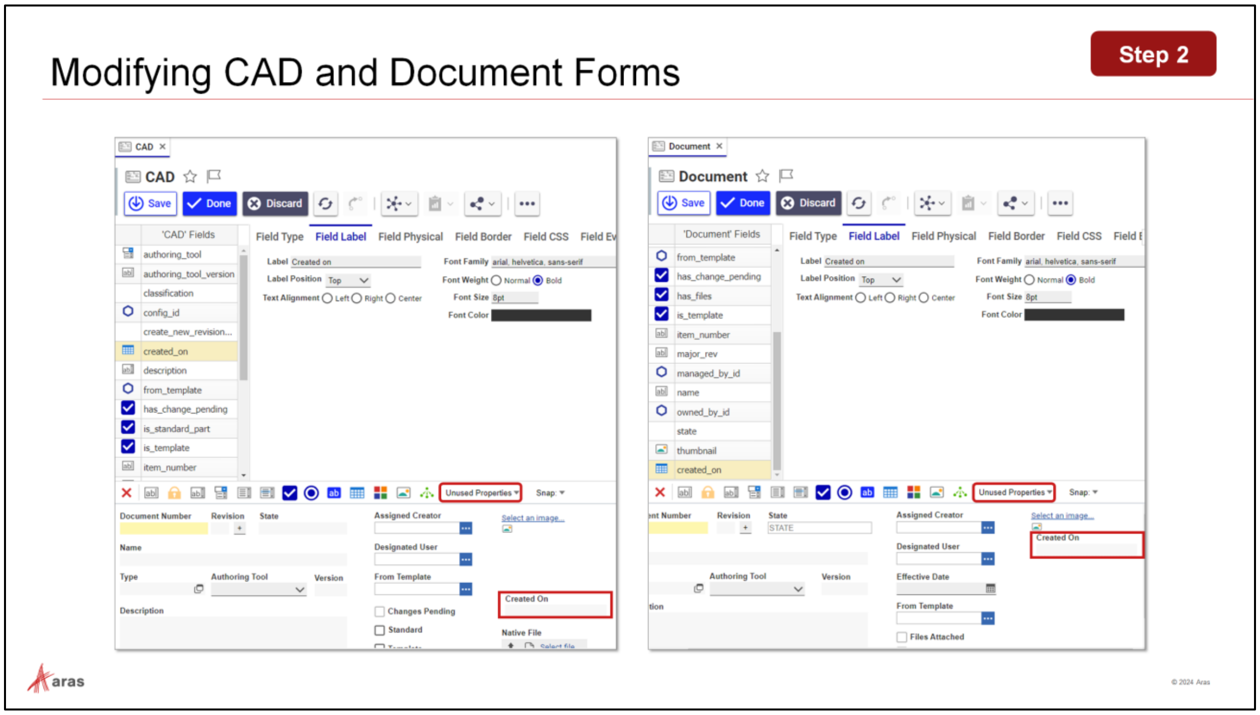
Try It … Export CAD and Document Forms Before Changes

1. Enter the server URL.
2. Click on the **Login** button to open the Login browser dialog.
3. Select the database from the drop-down, enter your admin username and password, then click the **Login** button.
4. Set the destination of the export, you will find the exported files in that directory.
5. Refresh packages (click on the three dotted = ellipsis button).
6. Locate the **Form** package group in the package definition **com.aras.innovator.solution.PLM** and click on the text (if you check the box then all contents are selected in right box).
7. Select the CAD and Document forms.
8. Click the **Export** button.

Note: You will be asked if you want to create a new directory or if you want to overwrite, respectively. Simply accept the option to create the new directory.

Try It … Confirm results

1. Navigate to the created export folder, e.g., **C:\ArasProjects\project1\localExports\Export1**, review the contents of the **PLM** folder and open the manifest file created by the Export Utility.
2. Navigate to your Export Utility log folder, e.g., **C:\~\PackageImportExportUtilities\Export\log** and review the Export log.



Modifying CAD and Document Forms

Step 2

Label: Created on
Label Position: Top
Text Alignment: Left Right Center
Font Family: arial, helvetica, sans-serif
Font Weight: Normal [✓ Bold]
Font Size: 8pt
Font Color

Modifying CAD and Document Forms

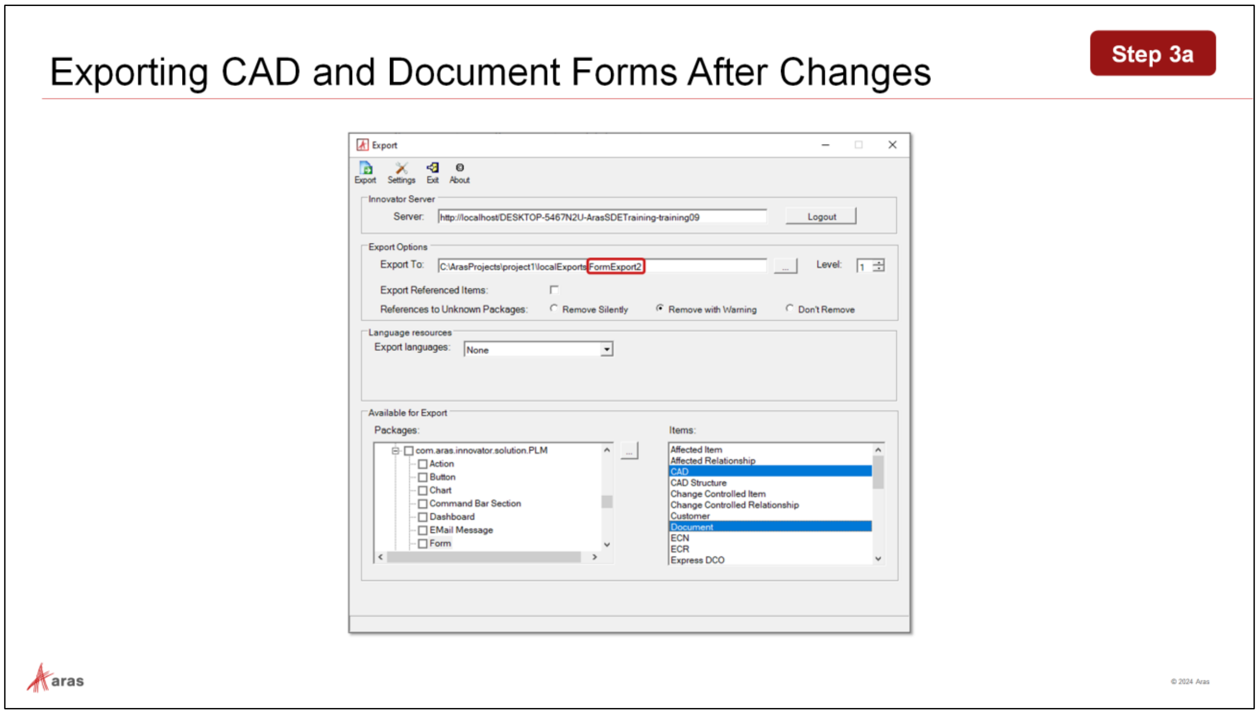
The changes shown in this example are the addition of an unused property into the forms for both ItemTypes.

Try It … Modify CAD Form

1. Login to Aras Innovator as user admin.
2. Select **Administration > Forms** and search for the **CAD** Form.
3. Switch to **Edit** mode.
4. Click on **Unused Properties** and select **created_on**.
5. Place the property on the form, change the label to **Created on** and save the form.

Try It … Modify Document Form

1. Login to Aras Innovator as user admin.
2. Select **Administration > Forms** and search for the **Document** Form.
3. Switch to **Edit** mode.
4. Click on **Unused Properties** and select **created_on**.
5. Place the property on the form, change the label to **Created on** and save the form.



Exporting CAD and Document Forms After Changes

Step 3a

Export

Innovator Server

Server: http://localhost/DESKTOP-5467N2U-ArasSDETraining-training09

Export Options

Export To: C:\ArasProjects\project1\localExports\FormExport2

Level: 1

Export Referenced Items: ☐

References to Unknown Packages: ○ Remove Silently [✓ Remove with Warning] ○ Don't Remove

Language resources

Export languages: None

Available for Export

Packages:

- ☐ com.aras.innovator.solution.PLM
  - ☐ Action
  - ☐ Button
  - ☐ Chart
  - ☐ Command Bar Section
  - ☐ Dashboard
  - ☐ EMail Message
  - ☐ Form

Items:

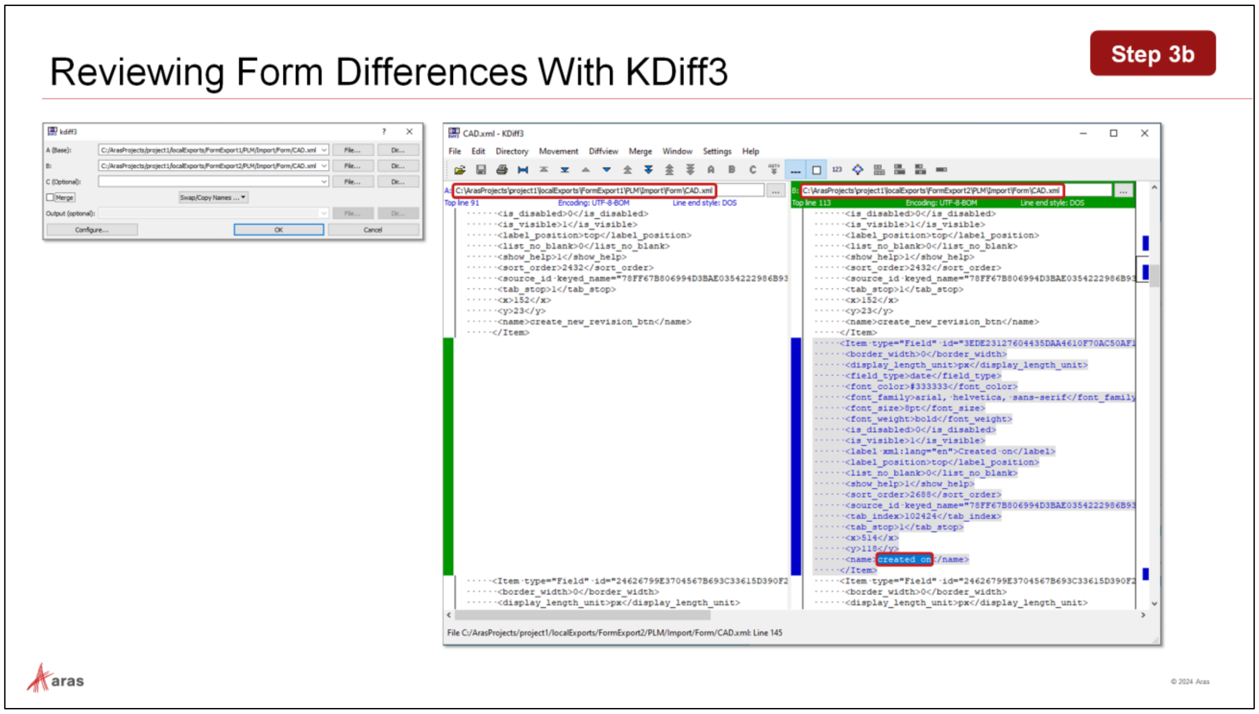
- Affected Item
- Affected Relationship
- [✓ CAD]
- CAD Structure
- Change Controlled Item
- Change Controlled Relationship
- Customer
- [✓ Document]
- ECN
- ECR
- Express DCO

Exporting CAD and Document Forms After Changes

Export your changes to the new destination folder C:\ArasProjects\project1\localExports\FormExport**2**, marking the two Form items CAD and Document.

Try It … Export CAD and Document Forms After Changes

1. Open the Export Utility in Administrator mode.
2. Populate the different fields as before and set the Export folder as **C:\ArasProjects\project1\localExports\FormExport2**.
3. Review the Export Utility log and the manifest file.



Reviewing Form Differences With KDiff3

Step 3b

kdiff3

A (Base): C:/ArasProjects/project1/localExports/FormExport1/PLM/Import/Form/CAD.xml

B: C:/ArasProjects/project1/localExports/FormExport2/PLM/Import/Form/CAD.xml

C (Optional):

Output (optional):

CAD.xml - KDiff3

A: C:\ArasProjects\project1\localExports\FormExport1\PLM\Import\Form\CAD.xml

Top line 91 Encoding: UTF-8-BOM Line end style: DOS

```
······<is_disabled>0</is_disabled>
······<is_visible>1</is_visible>
······<label_position>top</label_position>
······<list_no_blank>0</list_no_blank>
······<show_help>1</show_help>
······<sort_order>2432</sort_order>
······<source_id·keyed_name="78FF67B806994D3BAE0354222986B93
······<tab_stop>1</tab_stop>
······<x>152</x>
······<y>23</y>
······<name>create_new_revision_btn</name>
·····</Item>
·····<Item·type="Field"·id="24626799E3704567B693C33615D390F2
······<border_width>0</border_width>
······<display_length_unit>px</display_length_unit>
```

B: C:\ArasProjects\project1\localExports\FormExport2\PLM\Import\Form\CAD.xml

Top line 113 Encoding: UTF-8-BOM Line end style: DOS

```
······<is_disabled>0</is_disabled>
······<is_visible>1</is_visible>
······<label_position>top</label_position>
······<list_no_blank>0</list_no_blank>
······<show_help>1</show_help>
······<sort_order>2432</sort_order>
······<source_id·keyed_name="78FF67B806994D3BAE0354222986B93
······<tab_stop>1</tab_stop>
······<x>152</x>
······<y>23</y>
······<name>create_new_revision_btn</name>
·····</Item>
·····<Item·type="Field"·id="3EDE23127604435DAA4610F70AC50AF1
······<border_width>0</border_width>
······<display_length_unit>px</display_length_unit>
······<field_type>date</field_type>
······<font_color>#333333</font_color>
······<font_family>arial,·helvetica,·sans-serif</font_family
······<font_size>8pt</font_size>
······<font_weight>bold</font_weight>
······<is_disabled>0</is_disabled>
······<is_visible>1</is_visible>
······<label·xml:lang="en">Created·on</label>
······<label_position>top</label_position>
······<list_no_blank>0</list_no_blank>
······<show_help>1</show_help>
······<sort_order>2688</sort_order>
······<source_id·keyed_name="78FF67B806994D3BAE0354222986B93
······<tab_index>102424</tab_index>
······<tab_stop>1</tab_stop>
······<x>514</x>
······<y>118</y>
······<name>created_on</name>
·····</Item>
·····<Item·type="Field"·id="24626799E3704567B693C33615D390F2
······<border_width>0</border_width>
······<display_length_unit>px</display_length_unit>
```

File C:/ArasProjects/project1/localExports/FormExport2/PLM/Import/Form/CAD.xml: Line 145

Reviewing Form Differences With KDiff3

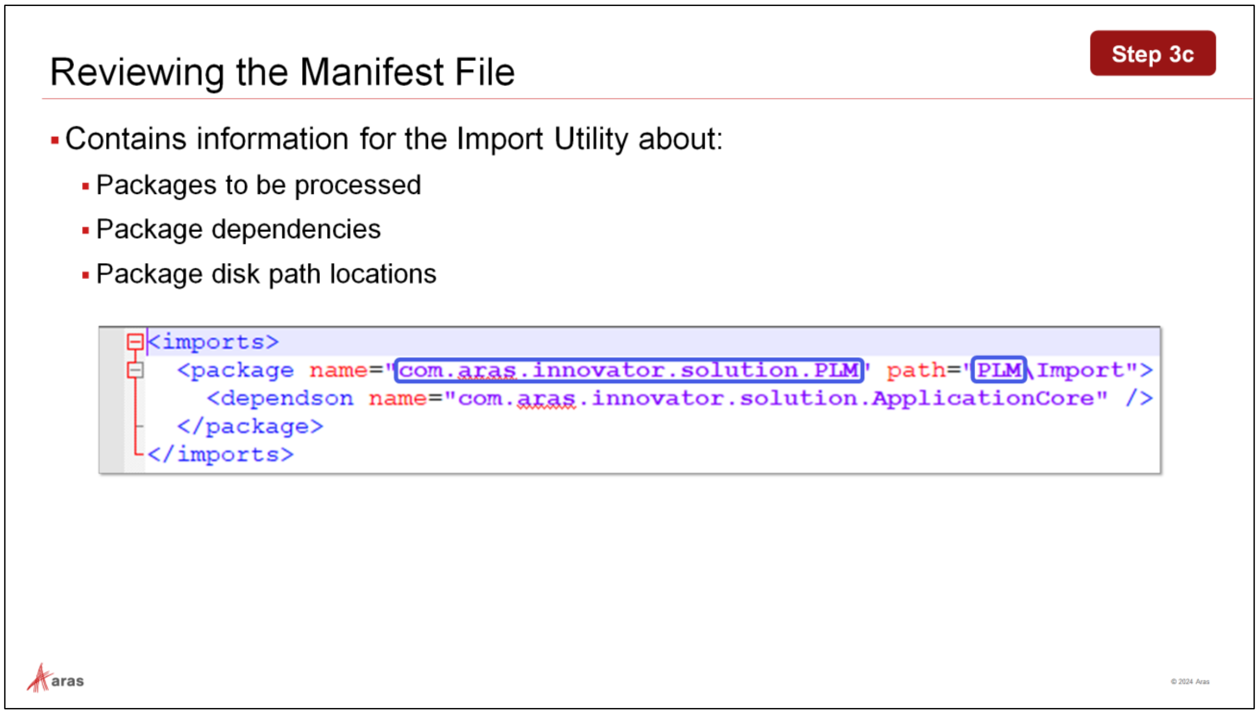
As a good exercise, analyze the differences between the export data before and after changes. We want to avoid situations where additional changes (currently not relevant) are present before we make a commit.

Try It … Review Differences in the Result Files for the CAD Form

1. Use KDiff3.exe to compare the two result files containing the AML for importing the CAD Form.
2. Select C:\ArasProjects\project1\localExports\FormExport1\PLM\Import\Form\CAD.xml as the **A (Base)** file.
3. Select C:\ArasProjects\project1\localExports\FormExport2\PLM\Import\Form\CAD.xml as the **B** file.
4. Observe the differences: We find the definition of one additional field with the name property **created_on** in the exported xml-file after the change – that reflects exactly what we changed from the UI.

Try It … Review Differences in the Result Files for the Document Form

1. Use KDiff3.exe to compare the two result files containing the AML for importing the Document Form.
2. Select C:\ArasProjects\project1\localExports\FormExport1\PLM\Import\Form\Document.xml as the **A (Base)** file.
3. Select C:\ArasProjects\project1\localExports\FormExport2\PLM\Import\Form\Document.xml as the **B** file.
4. Observe the differences: there are no differences. We find the definition of one additional field with the name property **created_on** in the exported xml-file after the change – that reflects exactly what we changed from the UI.



Reviewing the Manifest File

Step 3c

- Contains information for the Import Utility about:
  - Packages to be processed
  - Package dependencies
  - Package disk path locations

```
<imports>
  <package name="com.aras.innovator.solution.PLM" path="PLM\Import">
    <dependson name="com.aras.innovator.solution.ApplicationCore" />
  </package>
</imports>
```

© 2024 Aras

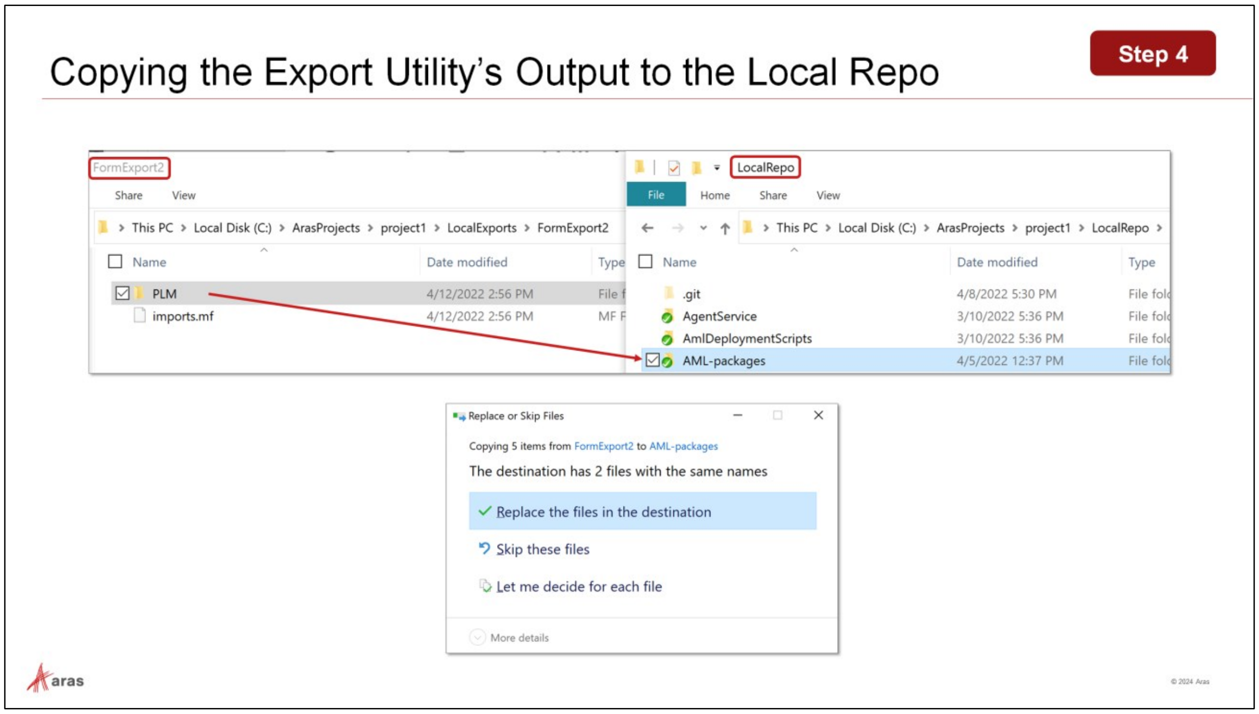
Reviewing the Manifest File

The Export Utility creates a manifest file that includes the exported package, i.e., **PLM** in our use case. The contents of the **imports.mf** file are already part of the current manifest file in the repository AML-package folder.

Try It … Compare Manifest Files in Export and Repo Folders

1. Open the first **imports.mf** file from C:\ArasProjects\project1\localExports\FormExport2 (= result of export).
2. Open the second **imports.mf** file from C:\ArasProjects\project1\LocalRepo\AML-packages (= used for future builds).
3. Compare the two files: Both files will be identical in the PLM section (you may also use Kdiff3 to compare files).

Note: the Export Utility simplifies the name of the package during the creation of folder names. Notice that the Export Utility only uses the last element of the package name **PLM** as folder name.



Copying the Export Utility's Output to the Local Repo

Step 4

FormExport2
Share
View
This PC > Local Disk (C:) > ArasProjects > project1 > LocalExports > FormExport2
Name
Date modified
Type
PLM
imports.mf
LocalRepo
File
Home
This PC > Local Disk (C:) > ArasProjects > project1 > LocalRepo
.git
AgentService
AmlDeploymentScripts
AML-packages
aras
© 2024 Aras

Replace or Skip Files

Copying 5 items from FormExport2 to AML-packages

The destination has 2 files with the same names

✓ Replace the files in the destination

Skip these files

Let me decide for each file

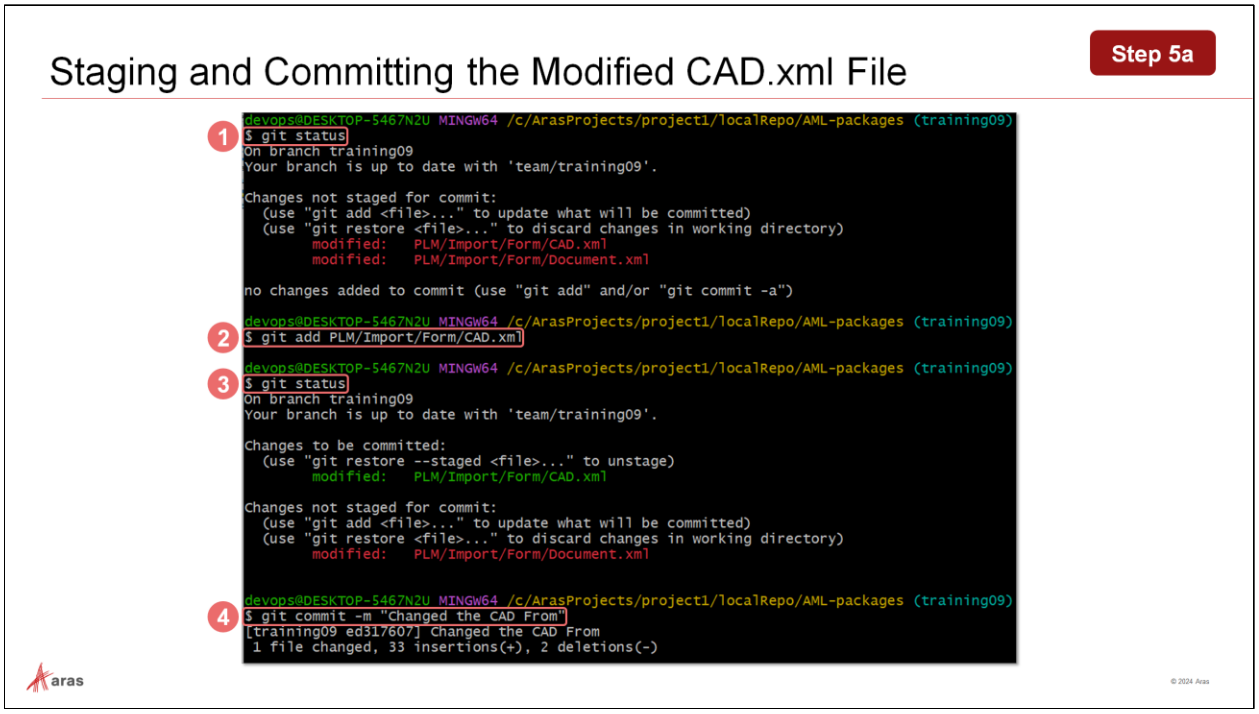
More details

Copying the Export Utility's Output to the Local Repo

When you have the export results, you can copy the top common folder and paste it in the repo local working directory and accept file replacement warnings. This approach allows you to avoid forgetting files.

Try It … Copy the Export Utility's Output to the Local Repo

1. Copy and paste the folder **C:\ArasProjects\project1\localExports\FormExport2\PLM** onto the corresponding folder in the local repo under AML-packages.
2. Accept the replacement and verify the contents of the **~\AML-packages\PLM\Import\Form** folder: the CAD.xml and Document.xml files should be the only updated files.



Staging and Committing the Modified CAD.xml File

Step 5a

```
devops@DESKTOP-5467N2U MINGW64 /c/ArasProjects/project1/localRepo/AML-packages (training09)
$ git status
On branch training09
Your branch is up to date with 'team/training09'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   PLM/Import/Form/CAD.xml
        modified:   PLM/Import/Form/Document.xml

no changes added to commit (use "git add" and/or "git commit -a")

devops@DESKTOP-5467N2U MINGW64 /c/ArasProjects/project1/localRepo/AML-packages (training09)
$ git add PLM/Import/Form/CAD.xml

devops@DESKTOP-5467N2U MINGW64 /c/ArasProjects/project1/localRepo/AML-packages (training09)
$ git status
On branch training09
Your branch is up to date with 'team/training09'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   PLM/Import/Form/CAD.xml

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   PLM/Import/Form/Document.xml


devops@DESKTOP-5467N2U MINGW64 /c/ArasProjects/project1/localRepo/AML-packages (training09)
$ git commit -m "Changed the CAD From"
[training09 ed317607] Changed the CAD From
 1 file changed, 33 insertions(+), 2 deletions(-)
```

Staging and Committing the Modified CAD.xml File

To add the modified file CAD.xml to the repository, we stage it by using the commands *git add …/CAD.xml* and *commit - m "commit message"*.

We do not stage the modified file Document.xml to see the different behavior.

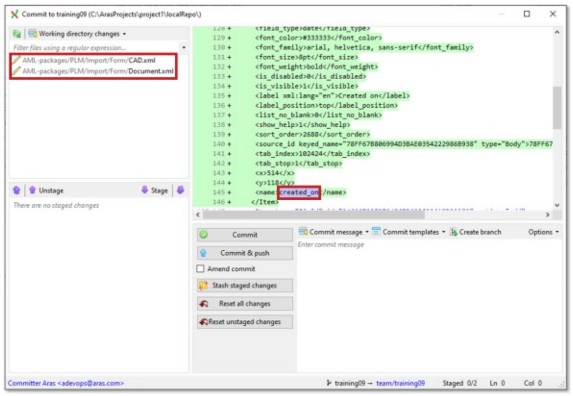
Try It … Stage and Commit the Modified CAD.xml File

1. Open Git Bash and navigate to **C:\ArasProjects\project1\LocalRepo\AML-packages**.
2. Verify the status of the Git repository by running *git status*.
   Result: you will see the two files **CAD.xml** and **Document.xml** as modified files but not staged files yet.
3. Stage the file **CAD.xml**: *git add PLM/Import/Form/CAD.xml*.
4. Verify now the new status of the Git repository by running *git status*.
   Result: you will see only the file **Document.xml** as not staged file; the file **CAD.xml** is displayed as a modified and staged file.
5. Commit the **CAD.xml** file using the command *git commit -m "Changed the CAD Form"*.

Note: You may also see the status of the Git repository before executing any steps within Git Extensions.

Confirming Your Changes in Git Extensions

Step 5b



Commit to training09 (C:\ArasProjects\project1\localRepo\)

Working directory changes

```
AML-packages/PLM/Import/Form/CAD.xml
AML-packages/PLM/Import/Form/Document.xml
```

```
 +        <field_type>date</field_type>
 +        <font_color>#333333</font_color>
 +        <font_family>arial, helvetica, sans-serif</font_family>
 +        <font_size>8pt</font_size>
 +        <font_weight>bold</font_weight>
 +        <is_disabled>0</is_disabled>
 +        <is_visible>1</is_visible>
 +        <label xml:lang="en">Created on</label>
 +        <label_position>top</label_position>
 +        <list_no_blank>0</list_no_blank>
 +        <show_help>1</show_help>
 +        <sort_order>2688</sort_order>
 +        <source_id keyed_name="78FF67B806994D3BAE0354222986B938" type="Body">78FF67
 +        <tab_index>102424</tab_index>
 +        <tab_stop>1</tab_stop>
 +        <x>514</x>
 +        <y>118</y>
 +        <name>created_on</name>
 +      </Item>
```

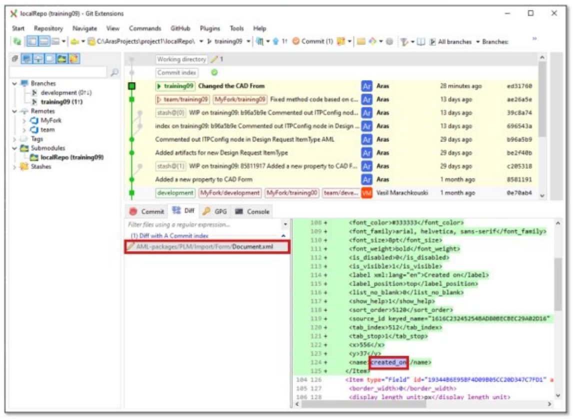
There are no staged changes

Enter commit message

Committer Aras <adevops@aras.com>   training09 → team/training09   Staged 0/2   Ln 0   Col 0

Before Staging/Committing

After Staging/Committing



localRepo (training09) - Git Extensions

```
Working directory  1
Commit index
training09  Changed the CAD From                                            Aras                 28 minutes ago  ed31760
team/training09  MyFork/training09  Fixed method code based on c...         Aras                 13 days ago     ae26a5e
stash@{0}  WIP on training09: b96a5b9e Commented out ITPConfig nod...       Aras                 13 days ago     39c8a74
index on training09: b96a5b9e Commented out ITPConfig node in Design ...    Aras                 13 days ago     696543a
Commented out ITPConfig node in Design Request ItemType AML                 Aras                 29 days ago     b96a5b9
Added artifacts for new Design Request ItemType                             Aras                 29 days ago     be2f40b
stash@{1}  WIP on training09: 85811917 Added a new property to CAD F...     Aras                 29 days ago     c205318
Added a new property to CAD Form                                            Aras                 1 month ago     8581191
development  MyFork/development  MyFork/training00  team/deve...            Vasil Marachkouski   1 month ago     0e70ab4
```

Filter files using a regular expression...

(1) Diff with A Commit index

```
AML-packages/PLM/Import/Form/Document.xml
```

```
 +        <font_color>#333333</font_color>
 +        <font_family>arial, helvetica, sans-serif</font_family>
 +        <font_size>8pt</font_size>
 +        <font_weight>bold</font_weight>
 +        <is_disabled>0</is_disabled>
 +        <is_visible>1</is_visible>
 +        <label xml:lang="en">Created on</label>
 +        <label_position>top</label_position>
 +        <list_no_blank>0</list_no_blank>
 +        <show_help>1</show_help>
 +        <sort_order>5120</sort_order>
 +        <source_id keyed_name="1616C232452548AD808EC8EC29A02D16"
 +        <tab_index>512</tab_index>
 +        <tab_stop>1</tab_stop>
 +        <x>556</x>
 +        <y>37</y>
 +        <name>created_on</name>
 +      </Item>
        <Item type="Field" id="19344B6E958F4D09B05CC20D347C7FD1" a
         <border_width>0</border_width>
         <display_length_unit>px</display_length_unit>
```

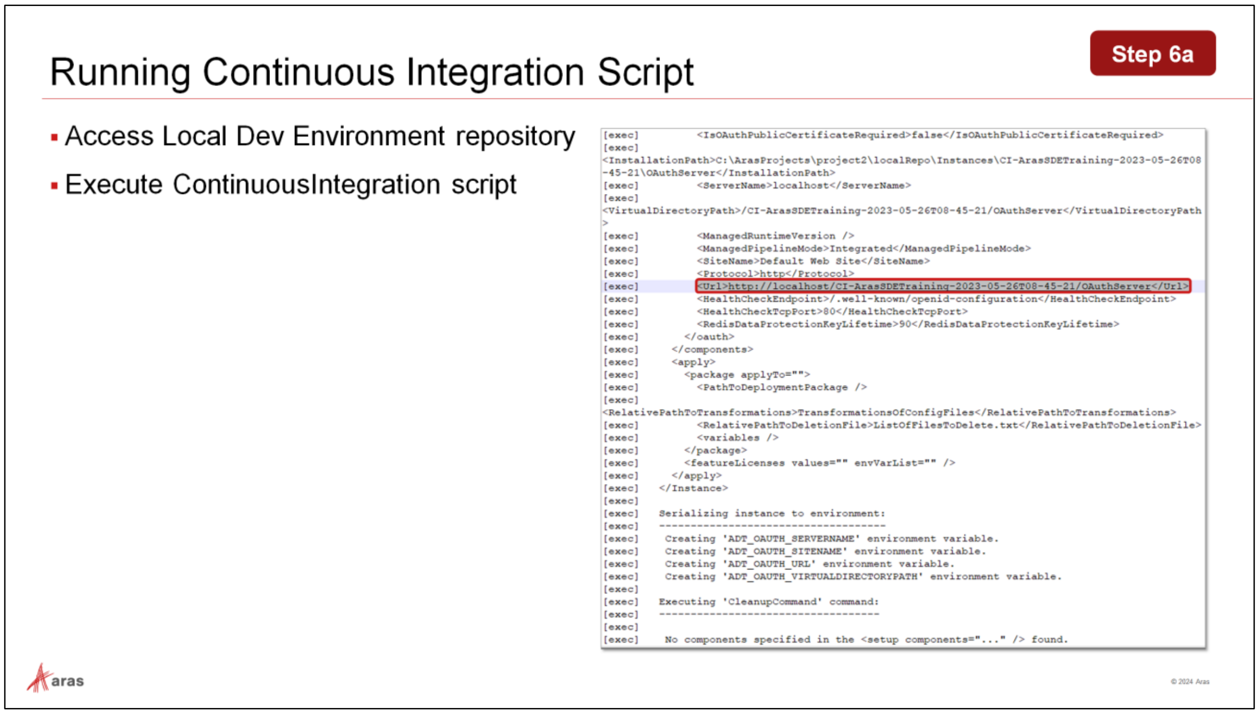
aras

Confirming Your Changes in Git Extensions

You may use Git Extensions to verify the status of your repo before and after staging and committing the changes.

Try It … Confirm Your Changes in Git Extensions

1. Open Git Extensions from the right-click context menu of your repository.
2. Click on **Commit (2)** in the toolbar to open the **Commit …** dialog: you will see 2 non-staged files (Before Staging/Committing screenshot).
3. After you have staged and committed the CAD.xml file (done in previous step), go to the **Working Directory** in Git Extensions.
4. Click on the **Diff** tab and select the Document.xml file.
5. Notice that it displays an additional Item block with the name property **created_on** (After Staging/Committing screenshot).



Running Continuous Integration Script

Step 6a

- Access Local Dev Environment repository
- Execute ContinuousIntegration script

```
[exec]        <IsOAuthPublicCertificateRequired>false</IsOAuthPublicCertificateRequired>
[exec]
<InstallationPath>C:\ArasProjects\project2\localRepo\Instances\CI-ArasSDETraining-2023-05-26T08
-45-21\OAuthServer</InstallationPath>
[exec]        <ServerName>localhost</ServerName>
[exec]
<VirtualDirectoryPath>/CI-ArasSDETraining-2023-05-26T08-45-21/OAuthServer</VirtualDirectoryPath
>
[exec]        <ManagedRuntimeVersion />
[exec]        <ManagedPipelineMode>Integrated</ManagedPipelineMode>
[exec]        <SiteName>Default Web Site</SiteName>
[exec]        <Protocol>http</Protocol>
[exec]        <Url>http://localhost/CI-ArasSDETraining-2023-05-26T08-45-21/OAuthServer</Url>
[exec]        <HealthCheckEndpoint>/.well-known/openid-configuration</HealthCheckEndpoint>
[exec]        <HealthCheckTcpPort>80</HealthCheckTcpPort>
[exec]        <RedisDataProtectionKeyLifetime>90</RedisDataProtectionKeyLifetime>
[exec]      </oauth>
[exec]    </components>
[exec]    <apply>
[exec]      <package applyTo="">
[exec]        <PathToDeploymentPackage />
[exec]
<RelativePathToTransformations>TransformationsOfConfigFiles</RelativePathToTransformations>
[exec]        <RelativePathToDeletionFile>ListOfFilesToDelete.txt</RelativePathToDeletionFile>
[exec]        <variables />
[exec]      </package>
[exec]      <featureLicenses values="" envVarList="" />
[exec]    </apply>
[exec]  </Instance>
[exec]
[exec]  Serializing instance to environment:
[exec]  -----------------------------------
[exec]   Creating 'ADT_OAUTH_SERVERNAME' environment variable.
[exec]   Creating 'ADT_OAUTH_SITENAME' environment variable.
[exec]   Creating 'ADT_OAUTH_URL' environment variable.
[exec]   Creating 'ADT_OAUTH_VIRTUALDIRECTORYPATH' environment variable.
[exec]
[exec]  Executing 'CleanupCommand' command:
[exec]  ----------------------------------
[exec]
[exec]   No components specified in the <setup components="..." /> found.
```

aras

© 2024 Aras

Running Continuous Integration Script

Run the ContinuousIntegration.ps1 script as administrator to ensure that your repository is still in a valid working state. You should receive a green success message.

CI Defined

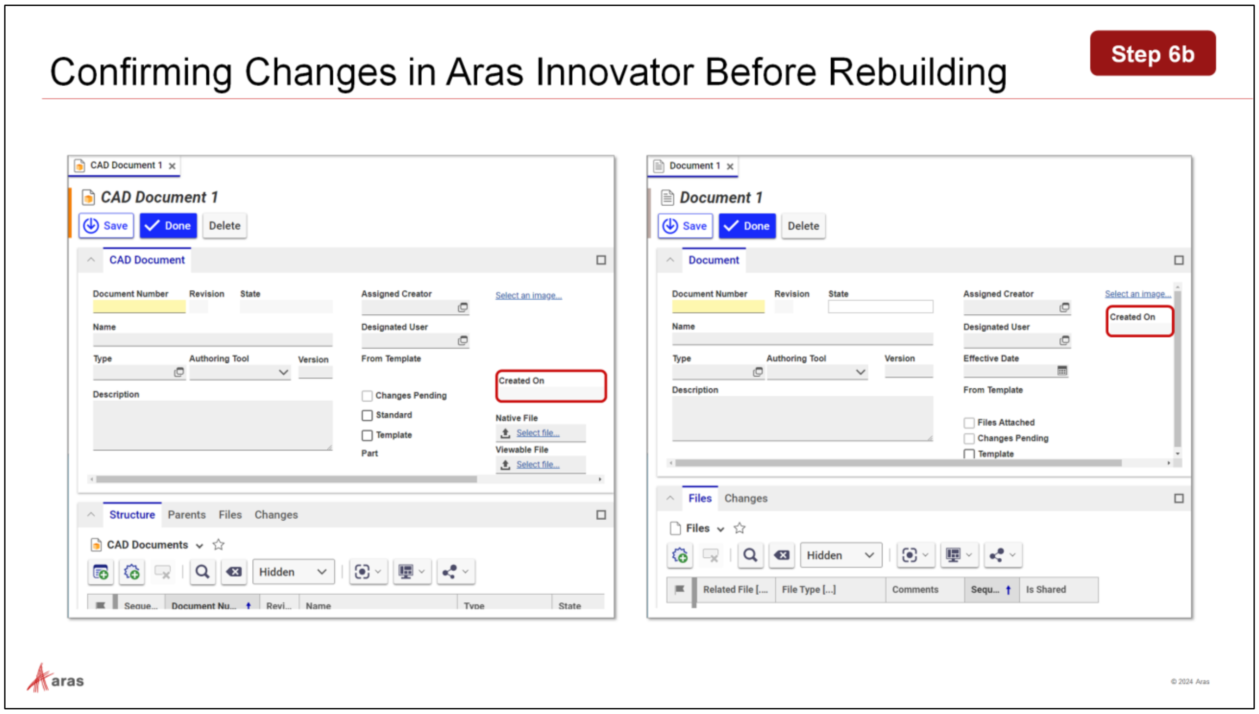
- An automated utility script is provided as part of each customer repository to perform final validation and verification that a build is successful.
- This script can be run by developers or system integrators manually to determine if the build passes or fails. Automation tools are also available to provide scheduled executions of this script on a dedicated CI server.

CI Steps

- The CI script runs all the integration tests that have been created for a project by installing and building a new instance of Aras Innovator, applying the project code and configuration, and making sure all tests are successful.
- A report indicates Success or Failure with a running log if issues need to be resolved. The script then deletes the running instance (and database).

Try It … Run Continuous Integration Script

1. Open an admin PowerShell session and navigate to the local repository at **C:\ArasProjects\project1\LocalRepo**.
2. Run script: .*ContinuousIntegration.ps1*.
3. Verify the result from the admin PowerShell window and by opening the log, i.e., **ContinuousIntegration.txt** in **~\localRepo\AutomatedProceduresOutput\NAntOutput**.
4. You may also view more detailed information from the folder **~\localRepo\AutomatedProceduresOutput\Logs** and its subfolders, **CommitStage**, **DeployStage**, and **InstanceTestsStage**, which contain log details for the 3 stages of the script.


Confirming Changes in Aras Innovator Before Rebuilding

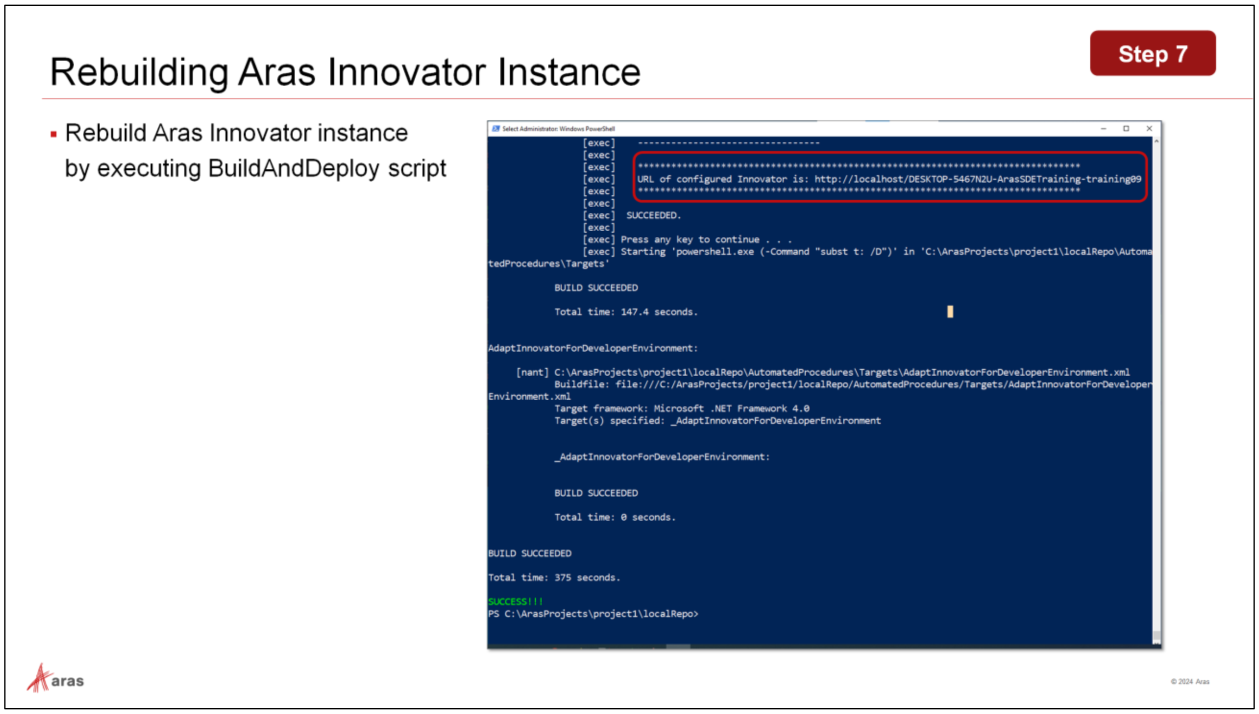
Step 6b

Confirming Changes in Aras Innovator Before Rebuilding

Verify that the current instance is using the modified forms for CAD and Document items. In the next steps we will delete the current Innovator instance and create a new one.

Try It … Confirm Changes in Aras Innovator Before Rebuilding

1. Go to your Innovator client in a browser.
2. Create a CAD document and notice the new field named **Created On**.
3. Create a document and notice the new field named **Created On**.



Rebuilding Aras Innovator Instance

Step 7

- Rebuild Aras Innovator instance by executing BuildAndDeploy script

Select Administrator: Windows PowerShell

```
                [exec]    ---------------------------------
                [exec]
                [exec]    ********************************************************************************
                [exec]    URL of configured Innovator is: http://localhost/DESKTOP-5467N2U-ArasSDETraining-training09
                [exec]    ********************************************************************************
                [exec]
                [exec]  SUCCEEDED.
                [exec]
                [exec] Press any key to continue . . .
                [exec] Starting 'powershell.exe (-Command "subst t: /D")' in 'C:\ArasProjects\project1\localRepo\Automa
tedProcedures\Targets'

           BUILD SUCCEEDED

           Total time: 147.4 seconds.


AdaptInnovatorForDeveloperEnvironment:

     [nant] C:\ArasProjects\project1\localRepo\AutomatedProcedures\Targets\AdaptInnovatorForDeveloperEnvironment.xml
           Buildfile: file:///C:/ArasProjects/project1/localRepo/AutomatedProcedures/Targets/AdaptInnovatorForDeveloper
Environment.xml
           Target framework: Microsoft .NET Framework 4.0
           Target(s) specified: _AdaptInnovatorForDeveloperEnvironment


           _AdaptInnovatorForDeveloperEnvironment:


           BUILD SUCCEEDED

           Total time: 0 seconds.


BUILD SUCCEEDED

Total time: 375 seconds.

SUCCESS!!!
PS C:\ArasProjects\project1\localRepo>
```

aras

© 2024 Aras

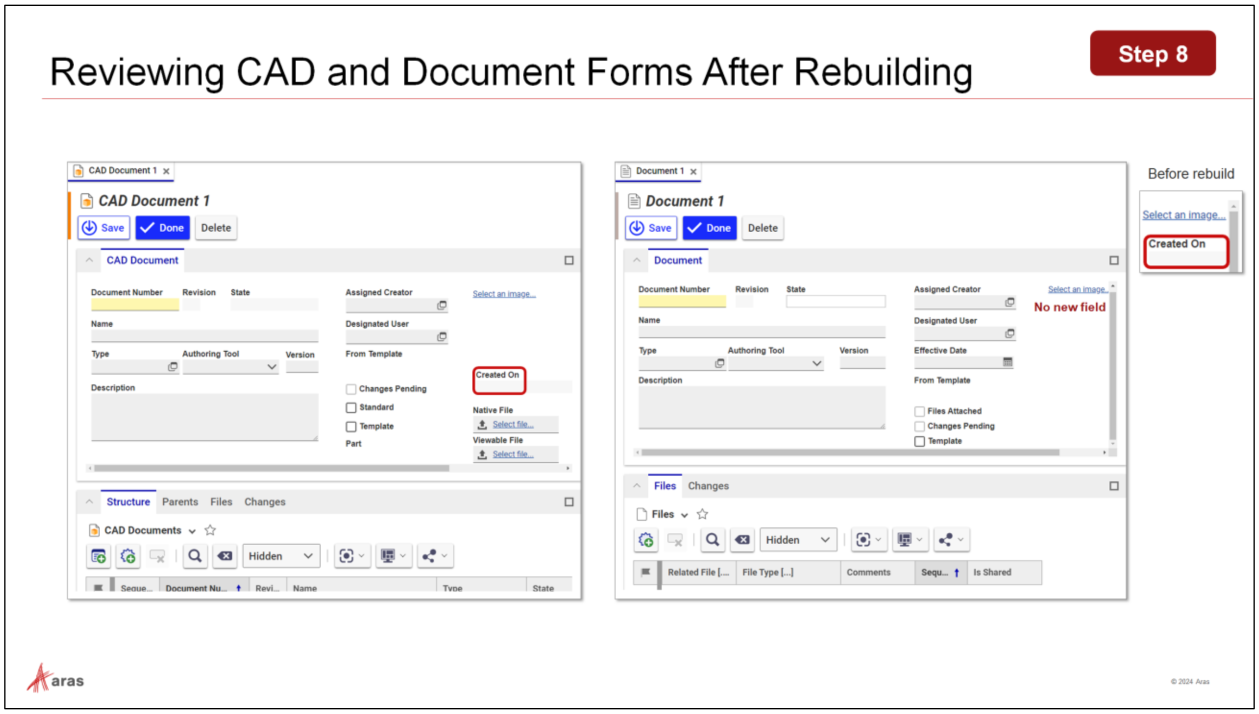
Rebuilding Aras Innovator Instance

Rebuild your Aras Innovator instance using the script **BuildAndDeploy.ps1**. You should receive a green success message.

In the output you will find the URL of the new Aras Innovator instance.

Try It … Rebuild Aras Innovator

1. Open an admin PowerShell session and navigate to the local repository at **C:\ArasProjects\project1\LocalRepo**.
2. Run script: *.\BuildAndDeploy.ps1*.
3. Verify the result from the admin PowerShell window and by opening the log, i.e., **BuildAndDeploy.txt** in **~\localRepo\AutomatedProceduresOutput\NAntOutput**.
4. You may also view more detailed information from the folder **~\localRepo\AutomatedProceduresOutput\Logs** and its subfolders, **CommitStage**, and **DeployStage**, which contain log details for the 2 stages of the script.



Reviewing CAD and Document Forms After Rebuilding

Step 8

Reviewing CAD and Document Forms After Rebuilding

The form for the CAD is like it was before the rebuild, meaning, it contains our intended change. On the other hand, the form for the document fell back to the initial state without a field for created_on.

The example for the document form illustrates what happens if you have not staged and committed your changes before the next rebuild.

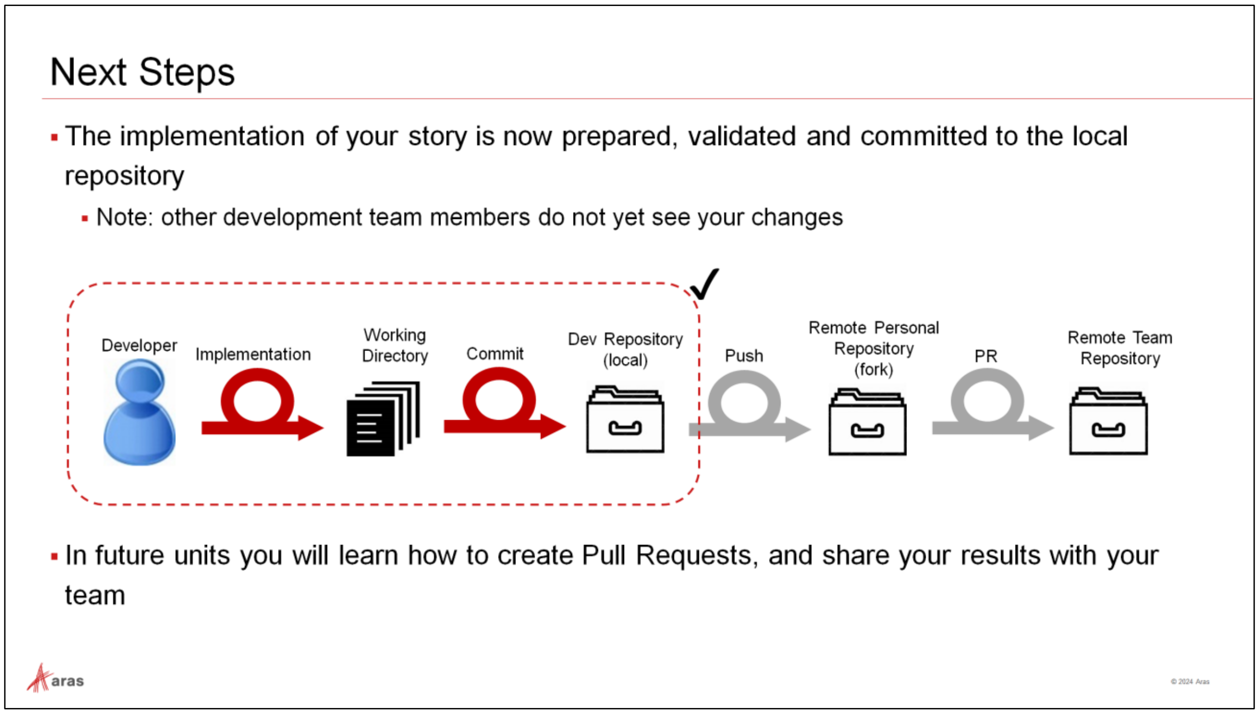
If you have kept your exports in separate folders, you can recover them.

Running **ContinuousIntegration.ps1** beforehand will not and cannot indicate the issue – it simply does not know what you forgot in the staging.

Avoid this situation by staging or committing changes before building, i.e., before running **BuildAndDeploy.ps1**.

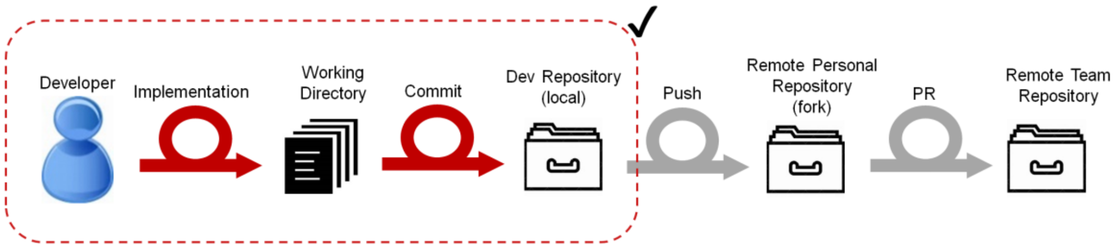
Try It … Review CAD and Document Forms After Rebuilding

1. Access the newly rebuilt Innovator instance.
2. Open the forms for the Document and the CAD document and notice any new changes.



Next Steps

- The implementation of your story is now prepared, validated and committed to the local repository
  - Note: other development team members do not yet see your changes



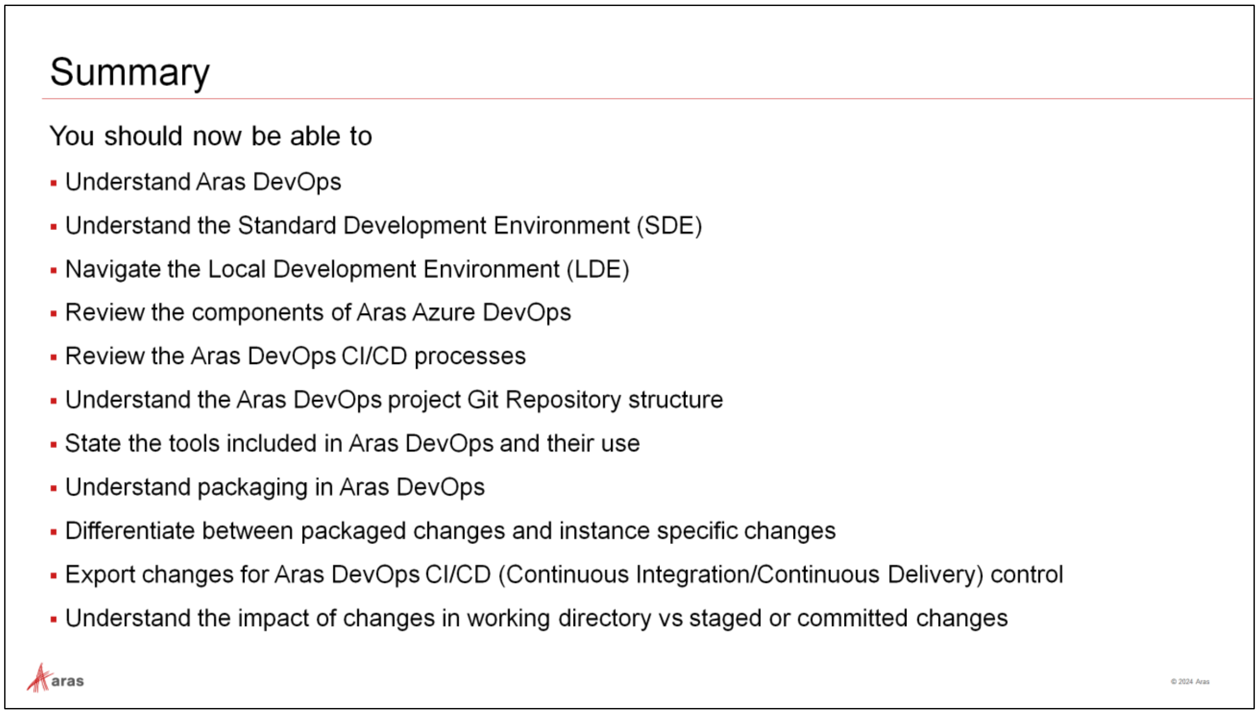
- In future units you will learn how to create Pull Requests, and share your results with your team

Next Steps

In the next unit we will learn how to create a Pull Request (PR) to share our contribution with the team.

Remember:

- Fetch and rebase frequently to get any new changes into the current local branch
- Running the BuildAndDeploy.ps1 pipeline creates a new Innovator Instance (overrides old, if it exists) and applies your latest changes from repo
- Developers frequently commit changes to their local repository



Summary

You should now be able to

- Understand Aras DevOps
- Understand the Standard Development Environment (SDE)
- Navigate the Local Development Environment (LDE)
- Review the components of Aras Azure DevOps
- Review the Aras DevOps CI/CD processes
- Understand the Aras DevOps project Git Repository structure
- State the tools included in Aras DevOps and their use
- Understand packaging in Aras DevOps
- Differentiate between packaged changes and instance specific changes
- Export changes for Aras DevOps CI/CD (Continuous Integration/Continuous Delivery) control
- Understand the impact of changes in working directory vs staged or committed changes

© 2024 Aras

Thank you for participating in this brief introduction to Aras DevOps. For more information, please go to the following web sites and pages:

- https://www.aras.com/en/why-aras/aras-enterprise-saas
- https://www.aras.com/community/subscriber-portal/training/w/development-best-practices/1003/aras-devops-training
- https://www.aras.com/community/DocumentationLibrary/ALL%20PDFs/DevOps/Aras%20DevOps%20-%20User%20Guide.pdf
- https://www.aras.com/community/documentationlibrary/DevOps/1.1/Content/StartPage/StartPage.htm
- https://www.aras.com/community/DocumentationLibrary/ALL%20PDFs/Flare%20PDF/Flare%20PDF/Aras%20Enterprise/Aras%20Enterprise%20Subscription%20(SaaS)%20-%20Administrator%20Guide.pdf