ACE 2024

STUDENT TRAINING GUIDE

Aras DevOps Packaging and Continuous Integration

aras

Revision MARCH 2024

Packaging and Continuous Integration

Overview

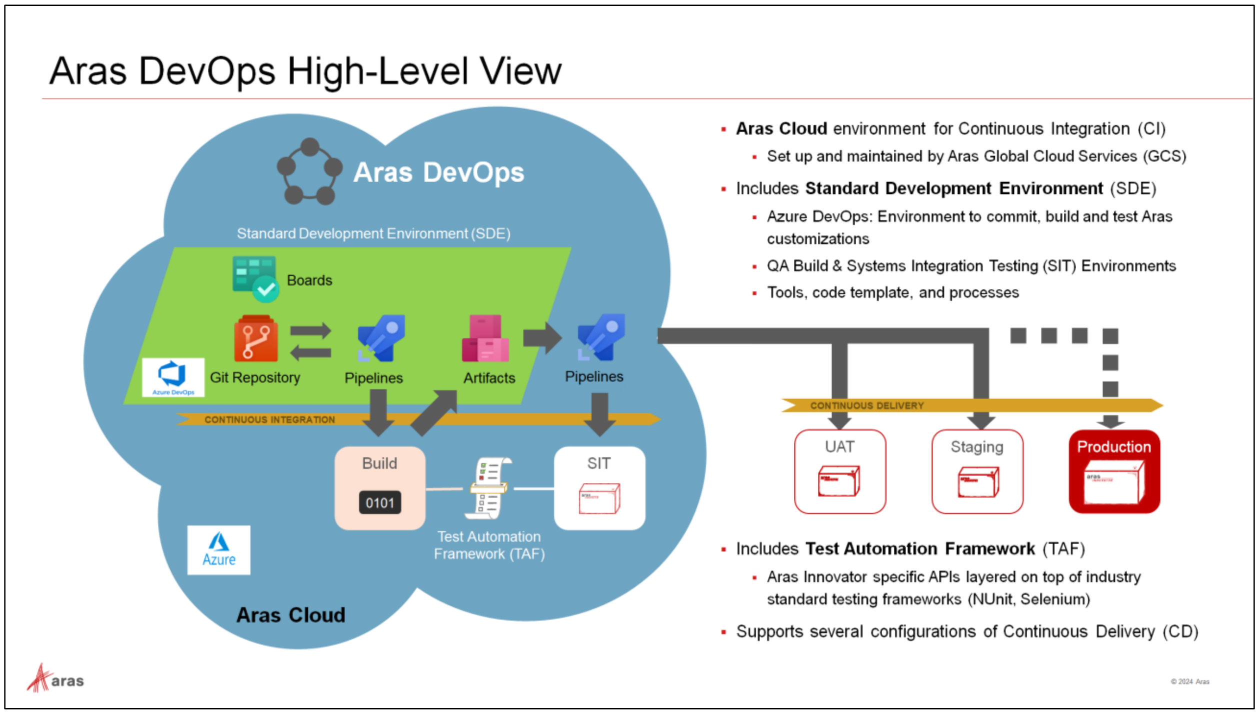
In this session, you will learn the basic guidelines for packaging items for Aras Innovator and how to use the packages within Aras DevOps Continuous Integration processes.

You will also build a simple project based on a use case, add all created items into the project package, export the project package into the customer repository, and commit the changes using Git (Source Control Management), after validation with the **ContinuousIntegration** pipeline.

Finally, you will rebuild your local Innovator instance with the project items with the **BuildAndDeploy** pipeline.

Objectives

- Understand Aras DevOps (AD)
- Understand packaging in Aras Innovator
- Differentiate between packaged changes and instance specific changes
- Understand packaging in Aras DevOps
- 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
- Build a sample project
- Package and export sample project
- Validate and commit sample project into local repository
- Rebuild local innovator instance with BuildAndDeploy pipeline



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

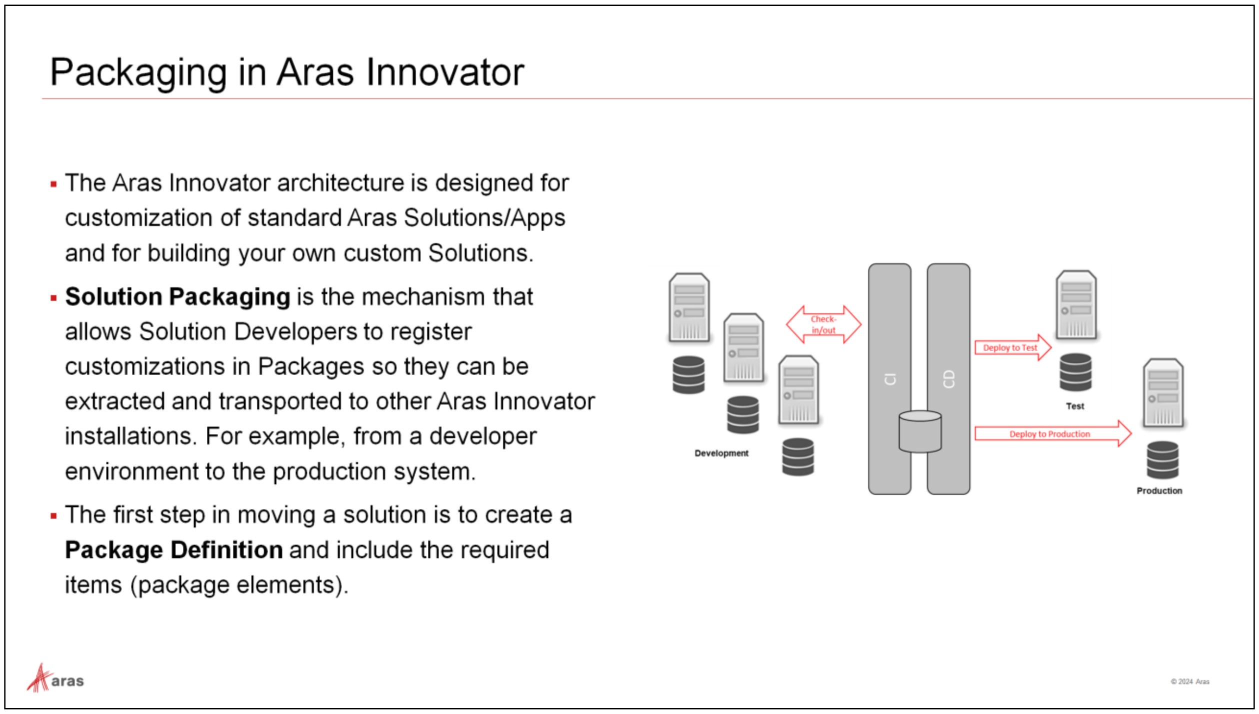
- Provides GIT for Source Control Management leveraging a Pull Request (PR) process.
- Provides Azure Pipelines for executing automated tests, code scans, etc.
- Provides Artifactory for storing all build artifacts to allow for reuse in deploying to multiple environments.
- Provides an SIT server to 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.

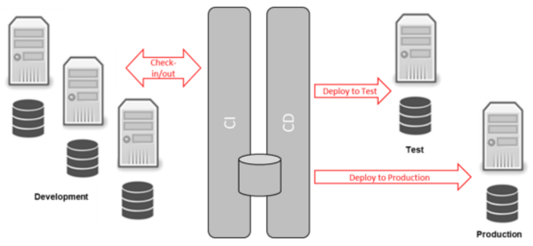
Standard Development Environment (SDE)

- 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.



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).



aras

© 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.

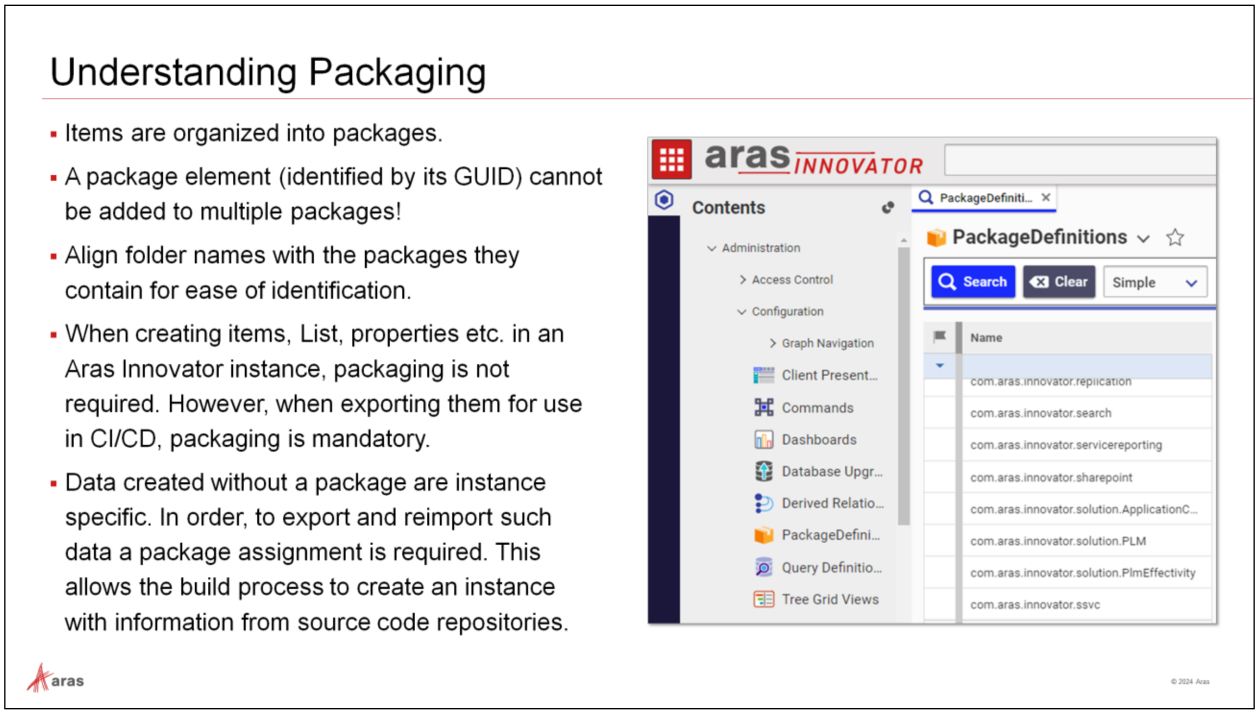
Making such changes directly in a business-critical system serving users is not good practice. As mentioned earlier, a central focus of DevOps is to instill the discipline of well-managed solution configurations, including change management and implementation.

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. In order, 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!

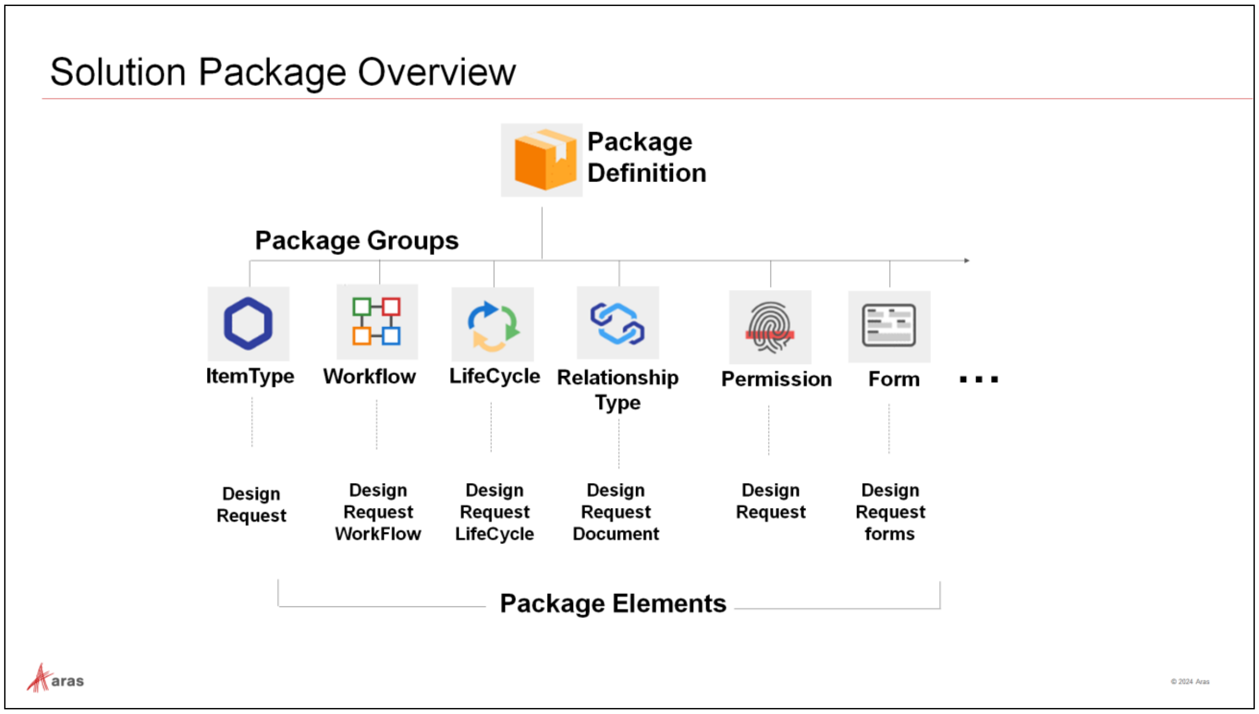
Try It … Explore existing Package Definitions

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!



Solution Package Overview

Solution Package Overview

The components of a Solution Package are outlined in this diagram. The number of Package Groups and Package Elements are dependent on an Aras Innovator release and can grow with each new release.

Package Definition

A Package Definition is a collection of Package Groups that contain Package Elements. For simple solutions you should be able to define a single package and add all relevant elements to it.

With larger solutions there can be cross dependencies between elements where one element of the same group may need to be created before the other. Or your solution is modular and has optional components. In this case your solution should be broken down into multiple packages.

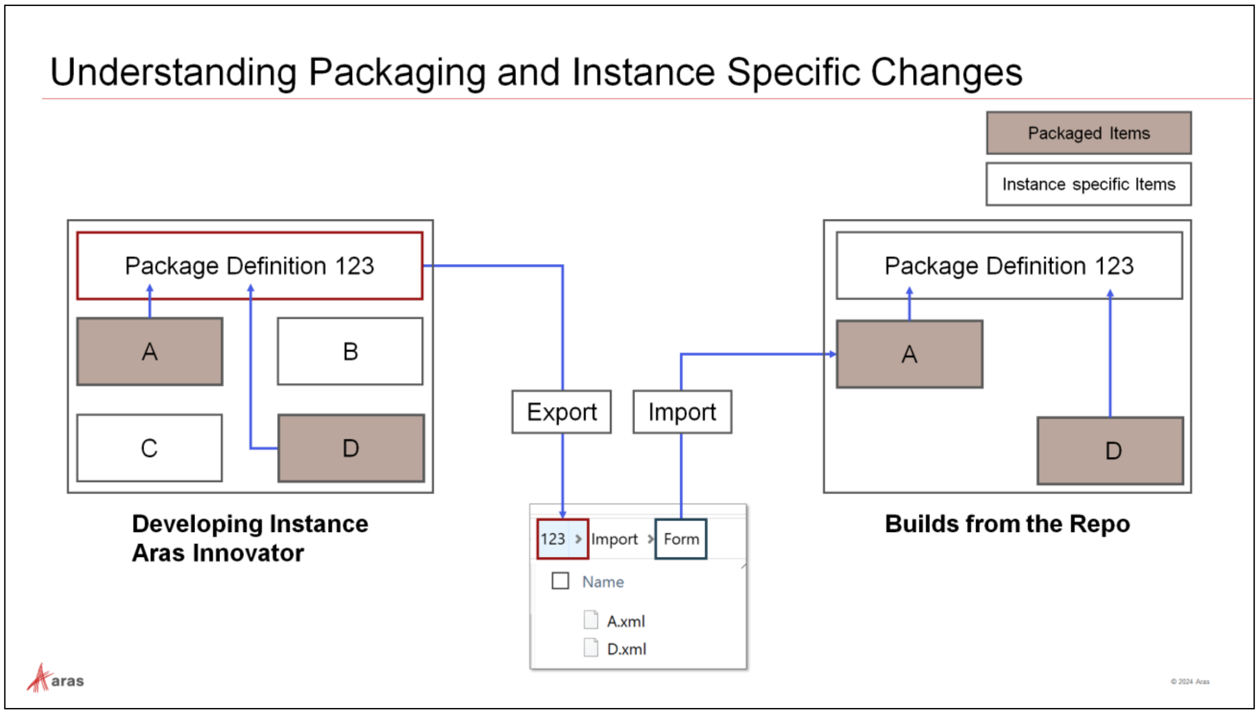
Package Definitions (in short, Packages) are exported from an Aras Innovator system they are defined in using the Packaging Utilities.

Package Groups

You do not need to create these. They are created automatically when adding an Element to a Package Definition. Names of Package Groups are built-in in the Modeling Engine. (like: Method, ItemType, List, RelationshipType, etc.)

Package Elements

In the packaging process Package Elements are added to a Package Definition. As an Aras Innovator Administrator, you will have buttons or menu actions “Add to Package Definition” to add selected elements to a package.



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.

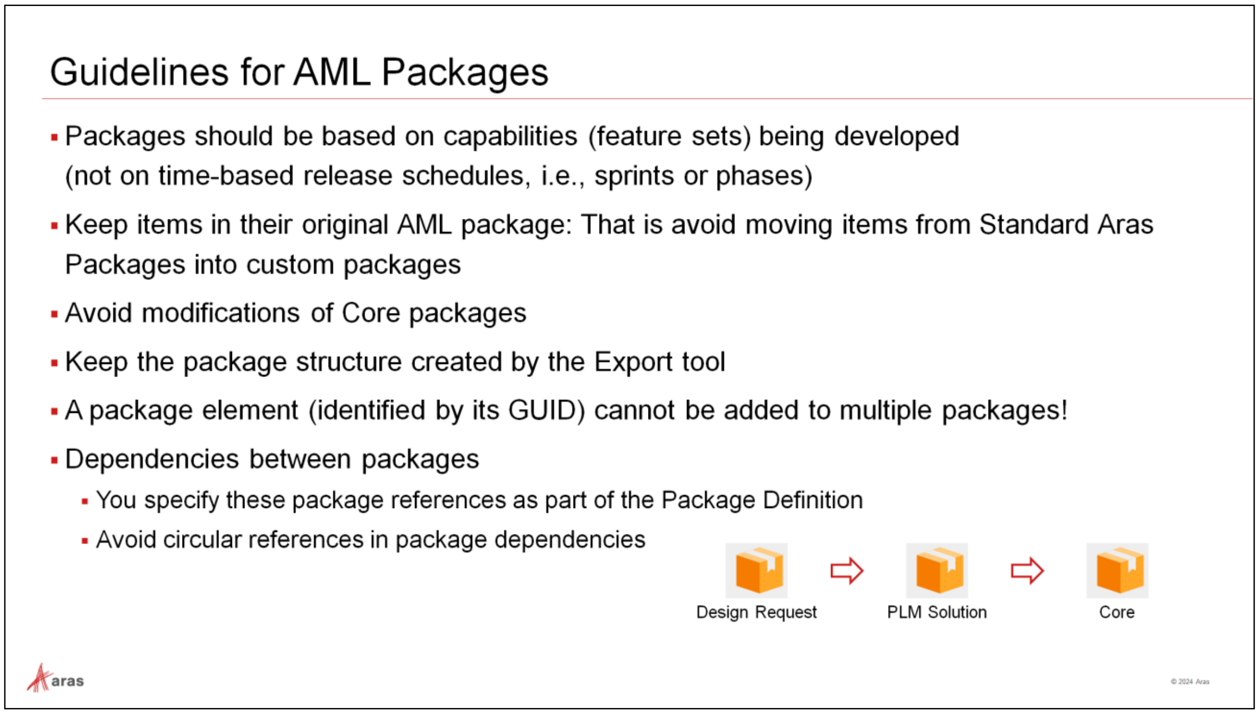
When changes are properly change-controlled then the corresponding solution configuration can be reproduced with assurance.

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.



Guidelines for AML Packages

- Packages should be based on capabilities (feature sets) being developed (not on time-based release schedules, i.e., sprints or phases)
- Keep items in their original AML package: That is avoid moving items from Standard Aras Packages into custom packages
- Avoid modifications of Core packages
- Keep the package structure created by the Export tool
- A package element (identified by its GUID) cannot be added to multiple packages!
- Dependencies between packages
  - You specify these package references as part of the Package Definition
  - Avoid circular references in package dependencies

Guidelines for AML Packages

Please do not modify package structure created by Import/Export tools. It might cause compatibilities issues for the new tools provided by Aras and issues for the support and future upgrades. Also, it will cause baselines synchronization issues.

When considering rules for AML Packages let us differentiate three different types of AML Packages:

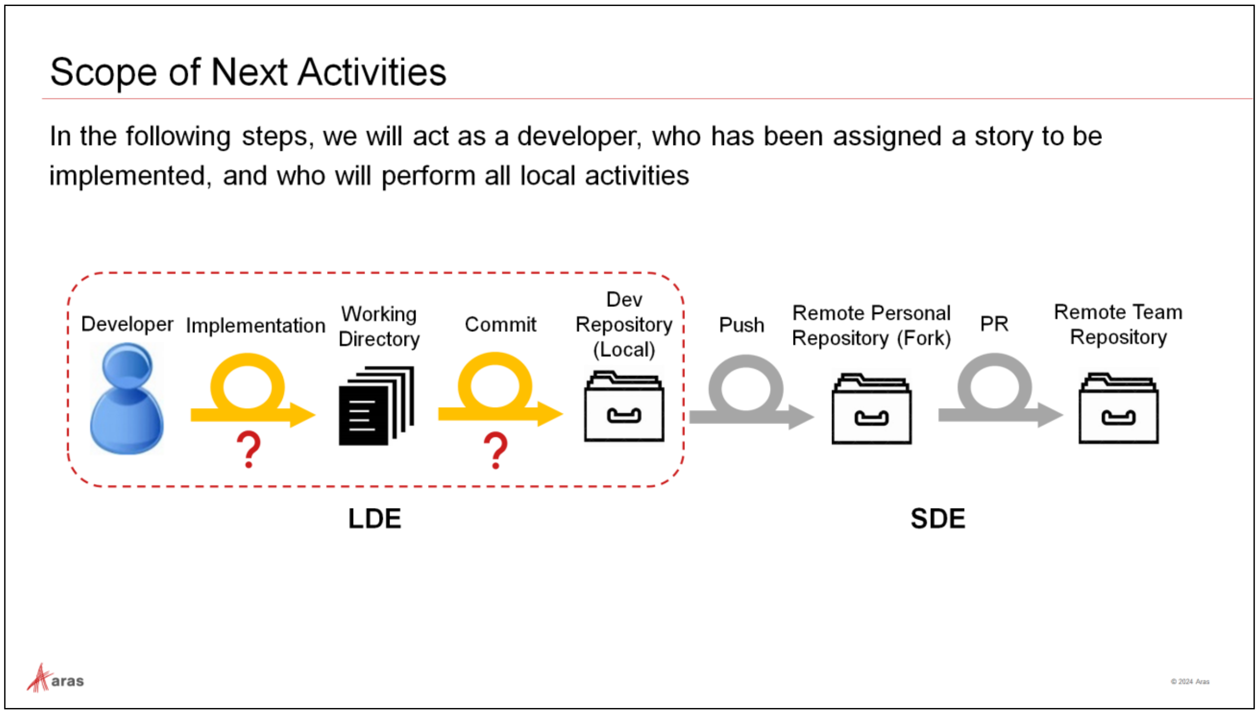
- **Custom Packages**
  Packages you create and manage, containing items that you created and that you manage.
- **Aras Solution Packages**
  Packages created and managed by Aras for a dedicated Solution. The names typically begin with **com.aras.innovator.solution**. Modify with care!
- **Aras Core Packages**
  Packages created and managed by Aras for the Core system.
  Do not modify, because any modifications (addition, change of value, deletion) put your Aras Innovator installation functionality at risk.

  The rules are meant to support you in keeping the functionality of your current Aras Innovator instance as well as reducing the risk of future collisions between changes you made, and changes made by Aras. Ignoring the rules can lead to future issues with support, upgrade, and baseline synchronization.

The packaging guidelines include keeping items in their original packages. Therefore, in our use case, we will change the CAD and Document forms within the Aras “com.aras.innovator.solution.PLM” solution package, which already exists in the baseline.

If you have a set of common items leveraged by multiple applications, which will force a circular dependency, it should be part of a Common Utility package loaded first.

Examples: Lists, Method, Conversion Rules, Permissions, Identities, some ItemTypes, etc., that could be used across multiple applications/packages.



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.

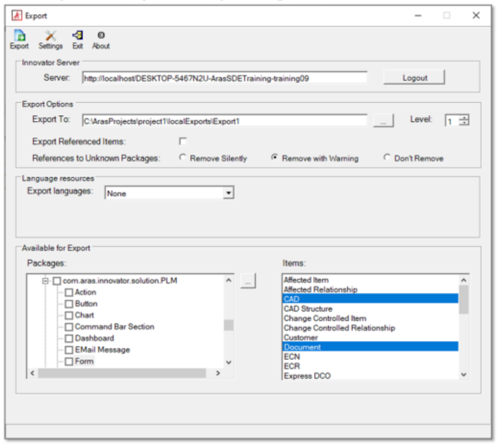
Work in SDE

- 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.

Exporting a Package

- The Export utility may be downloaded from: https://www.aras.com/en/support/downloads or obtained from the Aras Innovator CD image.
- It allows you to export DB packages and their elements into the Local Development Environment (LDE) file system.



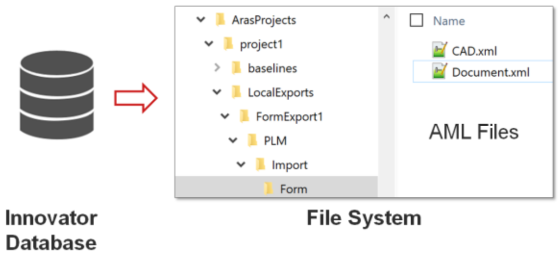
Innovator Server
Server: http://localhost/DESKTOP-5467N2U-ArasSDETraining-training09

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



aras

Exporting a Package

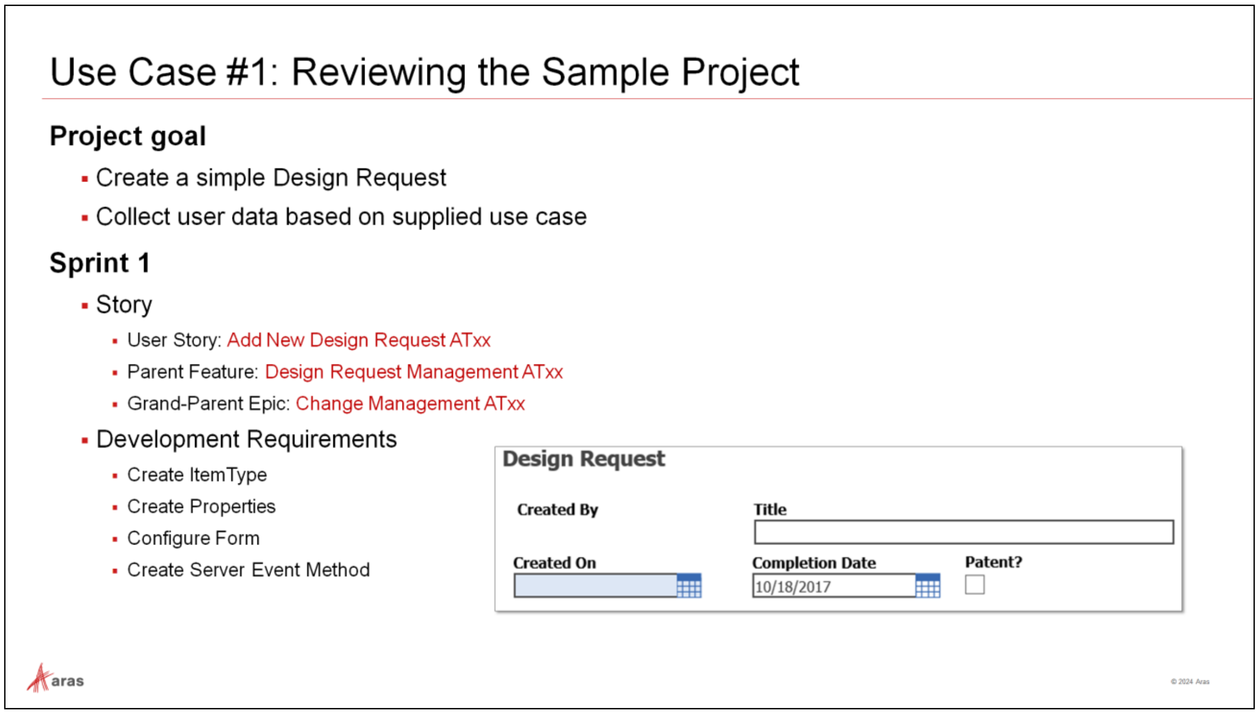
Once the Package Definition is complete, the package can be exported using the Export utility. This utility is a separate executable named export.exe that is available on the Aras Innovator CD image, or that may be downloaded from https://www.aras.com/en/support/downloads. It is important to select the Export utility belonging to the version and service pack of the installation from which you do the export.

The Export utility allows you to select a **Package Definition** from the database and create a package folder structure in the file system. Each **Package Group** (ItemType, Form, etc.) becomes a separate subfolder in the file. Within each subfolder, each **exported Item** is represented as an AML file with the same name as the exported Item.

In the example above, two Form definitions are exported and contained in the **Form** subdirectory. Note the remaining subfolder names – each represents a kind of exported Item from the database.

Try It … Open the Export Utility

1. Use Windows File Explorer and navigate to the place where the **export.exe** file has been stored.
   For example: C:\Users\devops\Desktop\Apps\PackageImportExportUtilities_R27\PackageImportExportUtilities (or as directed by instructor).
2. Double click the **export.exe** file to run the Export Utility.
3. In the **Server** field, enter the URL of the Aras Innovator Server (as provided by instructor).
4. Enter a valid administrator username and password in the respective fields of the browser that opens. In Aras classroom setting, use the **Username** of *admin* with the **Password** of *innovator*.
5. Click the **Login** button.



Use Case #1: Reviewing the Sample Project

Project goal

- Create a simple Design Request
- Collect user data based on supplied use case

Sprint 1

- Story
  - User Story: Add New Design Request ATxx
  - Parent Feature: Design Request Management ATxx
  - Grand-Parent Epic: Change Management ATxx
- Development Requirements
  - Create ItemType
  - Create Properties
  - Configure Form
  - Create Server Event Method

Design Request

Created By	Title	
Created On	Completion Date 10/18/2017	Patent?

Use Case #1: Reviewing the Sample Project

To demonstrate how the development process works, a small project has been designed, and this will allow users to collect information in a Design Request item.

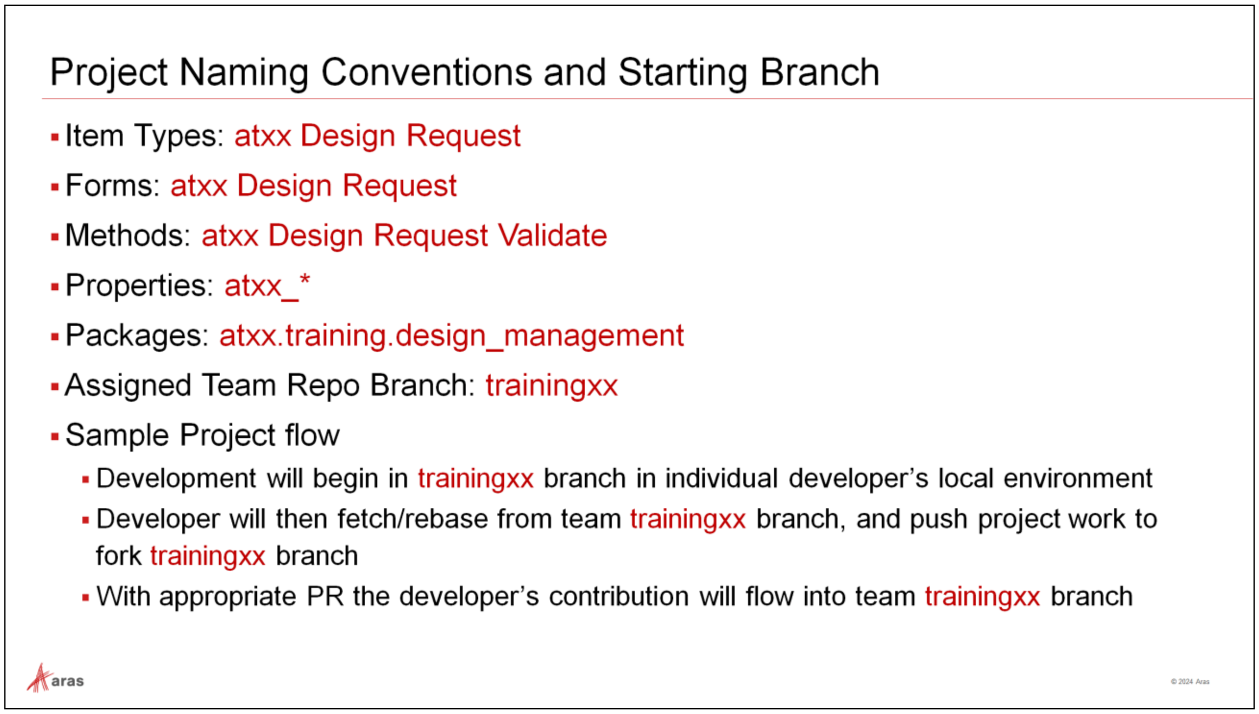
The first sprint should create the following:

- ItemType: Design Request
- Properties:
  - Request Number (Sequence)
  - Title (String)
  - Completion Date (Date)
  - Patent (Boolean)
- Form (Display properties for data entry)
- Method triggered by a Server Event (If Patent=1 the Completion Date is Required)

Requirement: The Design Request item should display the four properties on the form (custom and system properties). If the user checks the patent checkbox (Boolean) the completion date must be filled in to save the item.

The User Story has been created in a previous exercise, with the following hierarchy:

- User Story: **Add New Design Request ATxx** (xx: as provided by instructor)
- Parent Feature: **Design Request Management ATxx**
- Grand-Parent Epic: **Change Management ATxx**



Project Naming Conventions and Starting Branch

- Item Types: atxx Design Request
- Forms: atxx Design Request
- Methods: atxx Design Request Validate
- Properties: atxx_*
- Packages: atxx.training.design_management
- Assigned Team Repo Branch: trainingxx
- Sample Project flow
  - Development will begin in trainingxx branch in individual developer's local environment
  - Developer will then fetch/rebase from team trainingxx branch, and push project work to fork trainingxx branch
  - With appropriate PR the developer's contribution will flow into team trainingxx branch

Project Naming Conventions and Starting Branch

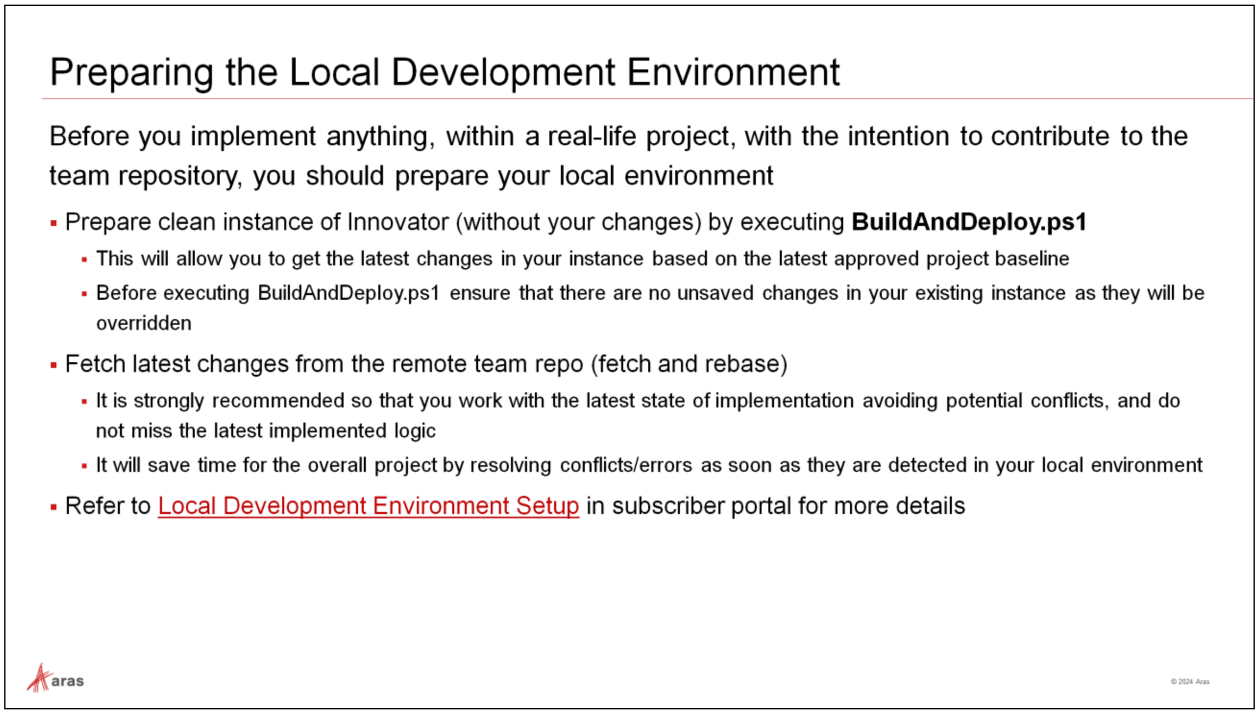
For the first sample, project development begins in the feature branch named **trainingxx**. (xx: as provided by instructor).

This branch was created in the customer repository in an earlier exercise. Make sure that the **trainingxx** branch is checked out before continuing.

Establish Naming Conventions

Naming conventions, including naming of the packages, is strongly recommended. Review Aras guidelines for naming conventions and adopt them as appropriate.

In our project, we will use the affix **atxx** (atxx: as provided by instructor), as either a prefix or a suffix.



Preparing the Local Development Environment

Before you implement anything, within a real-life project, with the intention to contribute to the team repository, you should prepare your local environment

- Prepare clean instance of Innovator (without your changes) by executing **BuildAndDeploy.ps1**
  - This will allow you to get the latest changes in your instance based on the latest approved project baseline
  - Before executing BuildAndDeploy.ps1 ensure that there are no unsaved changes in your existing instance as they will be overridden
- Fetch latest changes from the remote team repo (fetch and rebase)
  - It is strongly recommended so that you work with the latest state of implementation avoiding potential conflicts, and do not miss the latest implemented logic
  - It will save time for the overall project by resolving conflicts/errors as soon as they are detected in your local environment
- Refer to Local Development Environment Setup in subscriber portal for more details

© 2024 Aras

Preparing the Local Development Environment

You may go back to unit "Setting Up a Local Development Environment" to prepare your local environment for development.

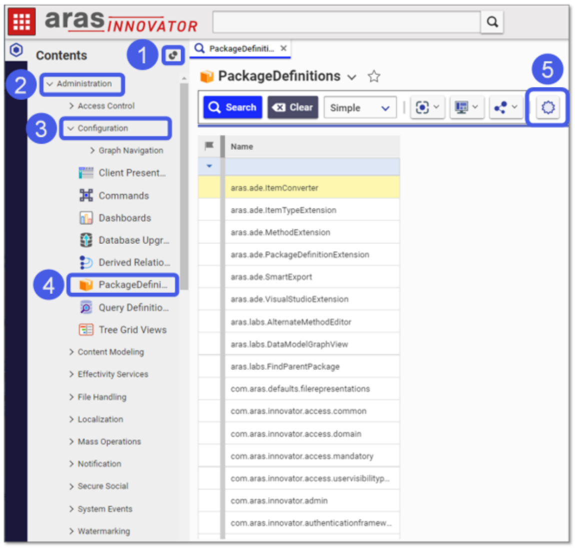
Recommendations

- Local Development Environment: should be prepared from the latest approved project baseline to ensure all the latest changes are included.
- Fetch and rebase
  - Should be performed multiple times throughout your development work, within the duration of the sprint, to ensure you have the latest changes that have been accepted in the team branch you are working on.
  - Helps minimize the number of conflicts/errors in your fork, and in the team repo; this also helps shortening the team's code review process.

Note: refer to Local Development Environment Setup in subscriber portal for more details.

Creating a Package

1. Pin up the Navigation Bar (TOC – Table of Contents)
2. Expand the Administration group
3. Expand the Configuration group
4. Click on the search icon at the right of the PackageDefinitions item
5. Click on the New PackageDefinition icon



Name
aras.ade.ItemConverter
aras.ade.ItemTypeExtension
aras.ade.MethodExtension
aras.ade.PackageDefinitionExtension
aras.ade.SmartExport
aras.ade.VisualStudioExtension
aras.labs.AlternateMethodEditor
aras.labs.DataModelGraphView
aras.labs.FindParentPackage
com.aras.defaults.filerepresentations
com.aras.innovator.access.common
com.aras.innovator.access.domain
com.aras.innovator.access.mandatory
com.aras.innovator.access.uservisibilityp...
com.aras.innovator.admin
com.aras.innovator.authenticationframew...

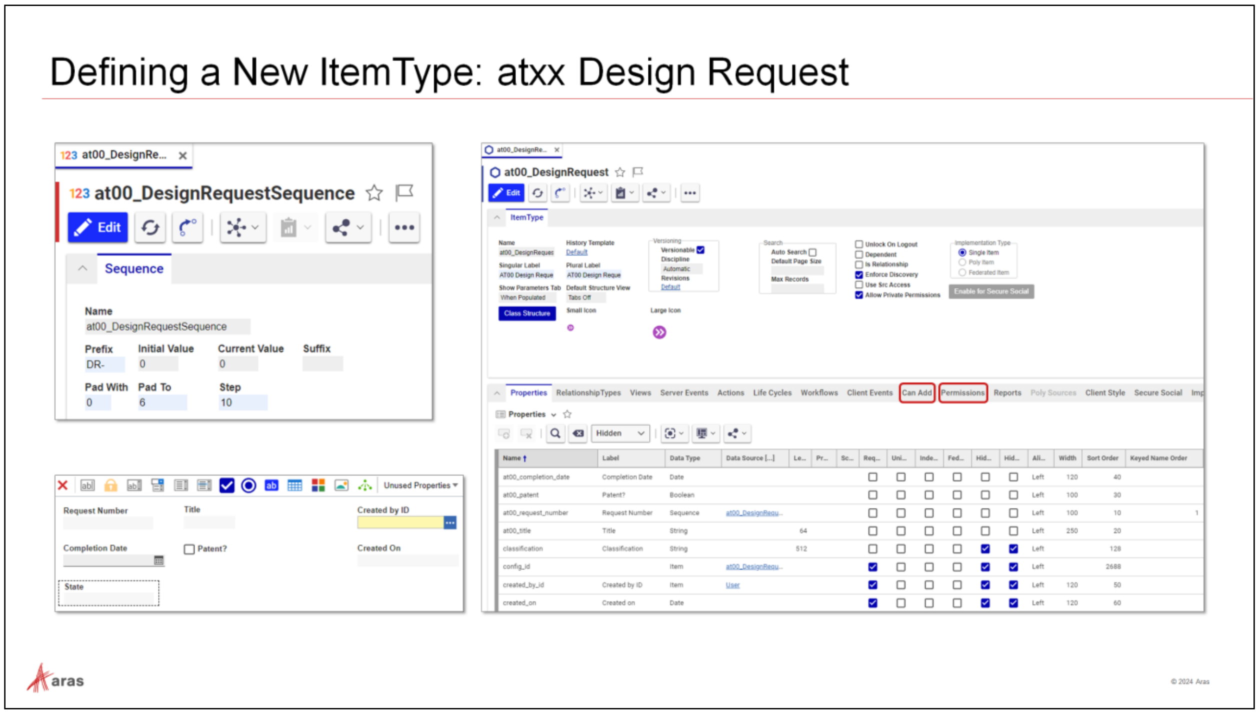
aras

Creating a Package

Follow the steps below to create a new package named *atxx.training.design_management* (atxx: as provided by instructor).

Try it … Create a Package

1. In Git Bash or Git Extensions ensure the **trainingxx** branch is checked out in the Local Development Environment.
2. In a browser window log in to your Innovator client.
3. Navigate to **Administration** > **Configuration** > **PackageDefinitions** and click the **New PackageDefinition** button.
4. Enter *atxx.training.design_management* for the **PackageDefinition Name** and click **Done**.



Defining a New ItemType: atxx Design Request

123 at00_DesignRequestSequence

Sequence

Name	at00_DesignRequestSequence		
Prefix	Initial Value	Current Value	Suffix
DR-	0	0	
Pad With	Pad To	Step	
0	6	10	

Request Number | Title | Created by ID
Completion Date | Patent? | Created On
State

at00_DesignRequest

ItemType

Name: at00_DesignReques
History Template: Default
Singular Label: AT00 Design Reque
Plural Label: AT00 Design Reque
Show Parameters Tab: When Populated
Default Structure View: Tabs Off
Versioning: Versionable ✓, Discipline: Automatic, Revisions: Default
Search: Auto Search ☐, Default Page Size, Max Records
☐ Unlock On Logout
☐ Dependent
☐ Is Relationship
✓ Enforce Discovery
☐ Use Src Access
✓ Allow Private Permissions
Implementation Type: ● Single Item ○ Poly Item ○ Federated Item

Properties | RelationshipTypes | Views | Server Events | Actions | Life Cycles | Workflows | Client Events | Can Add | Permissions | Reports | Poly Sources | Client Style | Secure Social

Name	Label	Data Type	Data Source [...]	Le...	Pr...	Sc...	Req...	Uni...	Inde...	Fed...	Hid...	Hid...	Ali...	Width	Sort Order	Keyed Name Order
at00_completion_date	Completion Date	Date					☐	☐	☐	☐	☐	☐	Left	120	40	
at00_patent	Patent?	Boolean					☐	☐	☐	☐	☐	☐	Left	100	30	
at00_request_number	Request Number	Sequence	at00_DesignRequ...				☐	☐	☐	☐	☐	☐	Left	100	10	1
at00_title	Title	String		64			☐	☐	☐	☐	☐	☐	Left	250	20	
classification	Classification	String		512			☐	☐	☐	☐	✓	✓	Left		128	
config_id		Item	at00_DesignRequ...				✓	☐	☐	☐	✓	✓	Left		2688	
created_by_id	Created by ID	Item	User				✓	☐	☐	☐	✓	✓	Left	120	50	
created_on	Created on	Date					✓	☐	☐	☐	✓	✓	Left	120	60	

© 2024 Aras

Defining a New Item Type: atxx Design Request

Follow the steps below to create a new **ItemType** named *atxx_DesignRequest* (atxx: as provided by instructor), with the following details:

- A new **Sequence**: *atxx_DesignRequestSequence* that will be used in a new property to number the design requests (DR-0010, DR-0020, etc.)
- New **Properties** to collect data about the request: *atxx_title (Title)*, *atxx_completion_date (Completion Date)*, *atxx_patent (Patent)*, and *atxx_request_number (Request Number)*
- **Permissions**: **Can Add** (World) and **Default Access**
- A new **Method**: *atxx_DesignRequestValidate* for the **OnBeforeAdd** and **OnBeforeUpdate** Server Events
- Assignment to the root of the **TOC** for quick access

Try it … Create a Sequence

1. Navigate to **Administration** > **Sequences** and create a new **Sequence**.
2. Name the sequence atxx_DesignRequestSequence.
3. Provide the following sequence values:

Prefix	DR-
Initial Value	0
Current Value	0
Suffix	
Pad With	0
Pad To	6
Step	10

4. Click the **Done** button to save the *atxx_DesignRequestSequence*.

Try it … Create the Design Request ItemType

1. Navigate to **Administration** > **ItemTypes** and create a new **ItemType**.
2. Enter or select the following values on the **ItemType** Form:

Name	atxx_DesignRequest
History Template	Default
Singular Label	ATxx Design Request
Plural Label	ATxx Design Requests
Show Parameters Tab	When Populated
Default Structure View	Tabs Off
Versionable	Checked
Small/Large Icons	Choose from provided image list

   Note: leave all other settings to default values
3. Click the **Save** button to save your new Design Request ItemType.
4. Assign the following permissions by clicking on each **Relationship** tab and adding a new row (**Add** button):

Relationship Tab	**Name Value**
Can Add	World (**Can Add** is checked)
Permissions	Default Access (**Is Default** is checked)

5. Click the **Save** button to save your new Design Request ItemType.

Try it … Create Custom Properties for the Design Request

1. Open the new **Design Request** ItemType in **Edit** mode (if necessary).
2. Select the **Relationship** tab labeled **Properties** and click the **New Property** button to add new properties.
3. Use the table below to provide the **Name**, **Label**, **Data Type**, and additional settings for each new property for the Design Request ItemType.

Property Name	**Label**	**Data Type**	**Length**	**Width**	**Sort**	**KNO**
atxx_title	Title	String	64	250	20	
atxx_completion_date	Completion Date	Date		120	40	
atxx_patent	Patent?	Boolean		100	30	
atxx_request_number	Request Number	Sequence		100	10	1

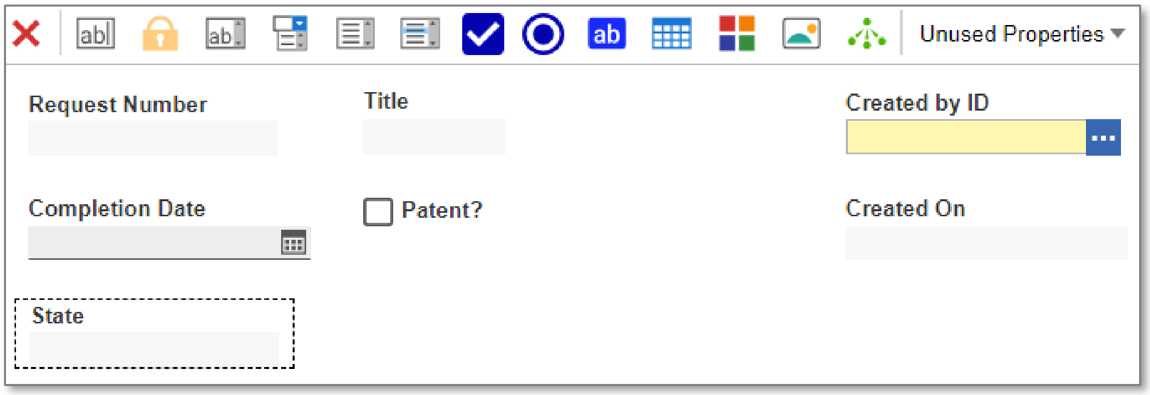
   Note: KNO is Keyed Name Order
4. Click the **Save** button to save your new Design Request ItemType.
5. Expose the following standard properties:

Property Name	**Label**	**Width**	**Sort**
created_by_id	Created by ID	120	50
created_on	Created on	120	60
state	State	100	70

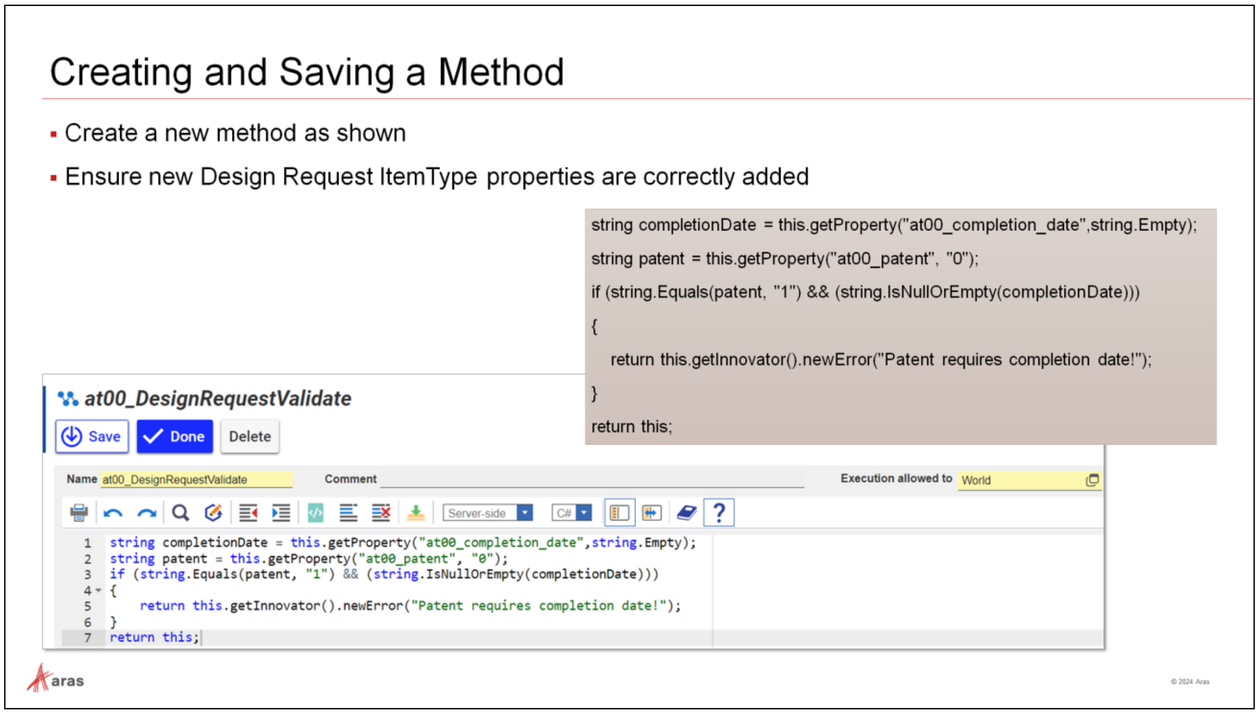
6. Click the **Done** button to save your new atxx_DesignRequest ItemType.

Try it … Update the New Design Request Form

1. Open the new **atxx_DesignRequest** ItemType.
2. Select the **Relationship** tab labeled **Views**, right-click on the **atxx_DesignRequest** form, and select **Actions > RebuildViewAction** from context menus.
3. Right-click again on the **atxx_DesignRequest** form and select **Open**.
4. Select **Edit** in the **atxx_DesignRequest** form , position the custom fields and standard fields as displayed in image:



5. Click the **Done** button to save changes to the new atxx_DesignRequest Form.



Creating and Saving a Method

- Create a new method as shown
- Ensure new Design Request ItemType properties are correctly added

```
string completionDate = this.getProperty("at00_completion_date",string.Empty);
string patent = this.getProperty("at00_patent", "0");
if (string.Equals(patent, "1") && (string.IsNullOrEmpty(completionDate)))
{
    return this.getInnovator().newError("Patent requires completion date!");
}
return this;
```

at00_DesignRequestValidate

Name: at00_DesignRequestValidate
Comment:
Execution allowed to: World
Server-side
C#

```
string completionDate = this.getProperty("at00_completion_date",string.Empty);
string patent = this.getProperty("at00_patent", "0");
if (string.Equals(patent, "1") && (string.IsNullOrEmpty(completionDate)))
{
    return this.getInnovator().newError("Patent requires completion date!");
}
return this;
```

Creating and Saving a Method

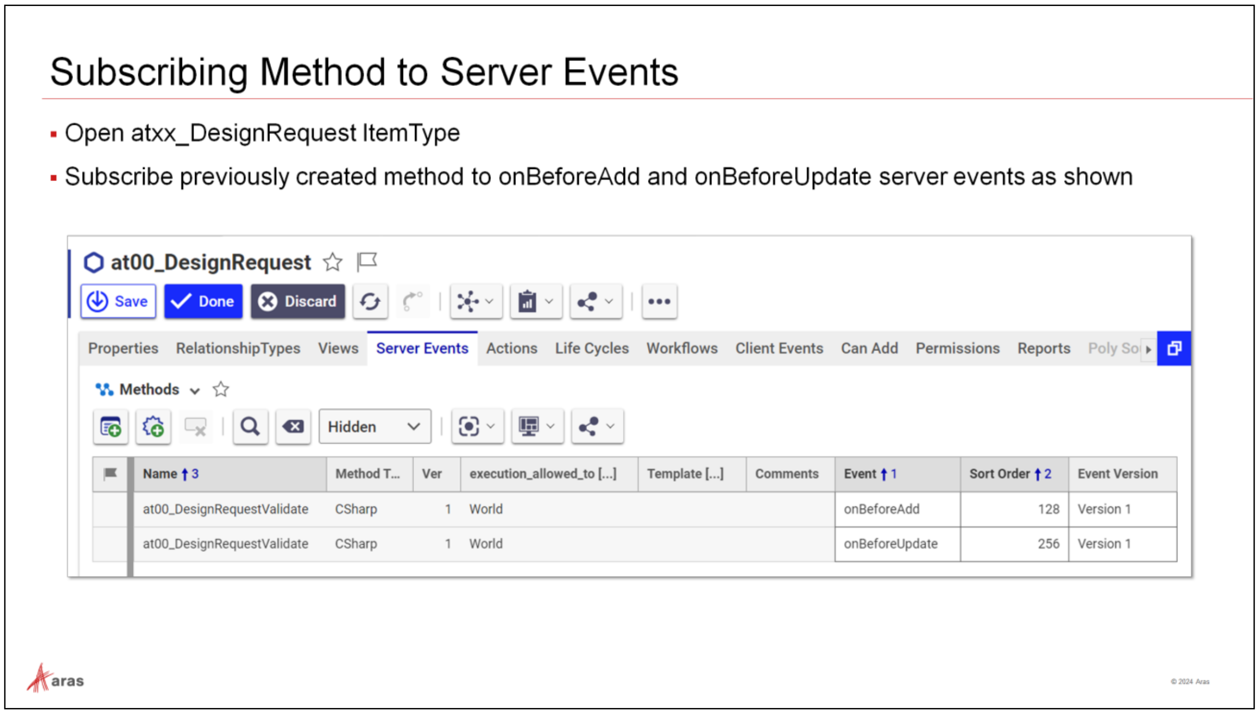
Follow the steps below to create a new **Method** named *atxx_DesignRequestValidate* (atxx: as provided by instructor).

Try It … Create and Save a Method

1. Navigate to **Administration** > **Methods** and click the **Create New Method** button.
2. Enter atxx_DesignRequestValidate for the **Name**.
3. Select **Server-side** and **C#** for the method basic properties.
4. Enter the following code:

```
string completionDate = this.getProperty("at00_completiondate",string.Empty);
string patent = this.getProperty("at00_patent", "0");
if (string.Equals(patent, "1") && (string.IsNullOrEmpty(completionDate)))
{
    return this.getInnovator().newError("Patent requires completion date!");
}
return this;
```

5. Ensure the **Execution allowed to** field is set to **World**.
6. Click the **Done** button to save the new method.



Subscribing Method to Server Events

- Open atxx_DesignRequest ItemType
- Subscribe previously created method to onBeforeAdd and onBeforeUpdate server events as shown

at00_DesignRequest

Save | Done | Discard

Properties | RelationshipTypes | Views | Server Events | Actions | Life Cycles | Workflows | Client Events | Can Add | Permissions | Reports | Poly So

Methods

Hidden

Name ↑3	Method T...	Ver	execution_allowed_to [...]	Template [...]	Comments	Event ↑1	Sort Order ↑2	Event Version
at00_DesignRequestValidate	CSharp	1	World			onBeforeAdd	128	Version 1
at00_DesignRequestValidate	CSharp	1	World			onBeforeUpdate	256	Version 1

aras

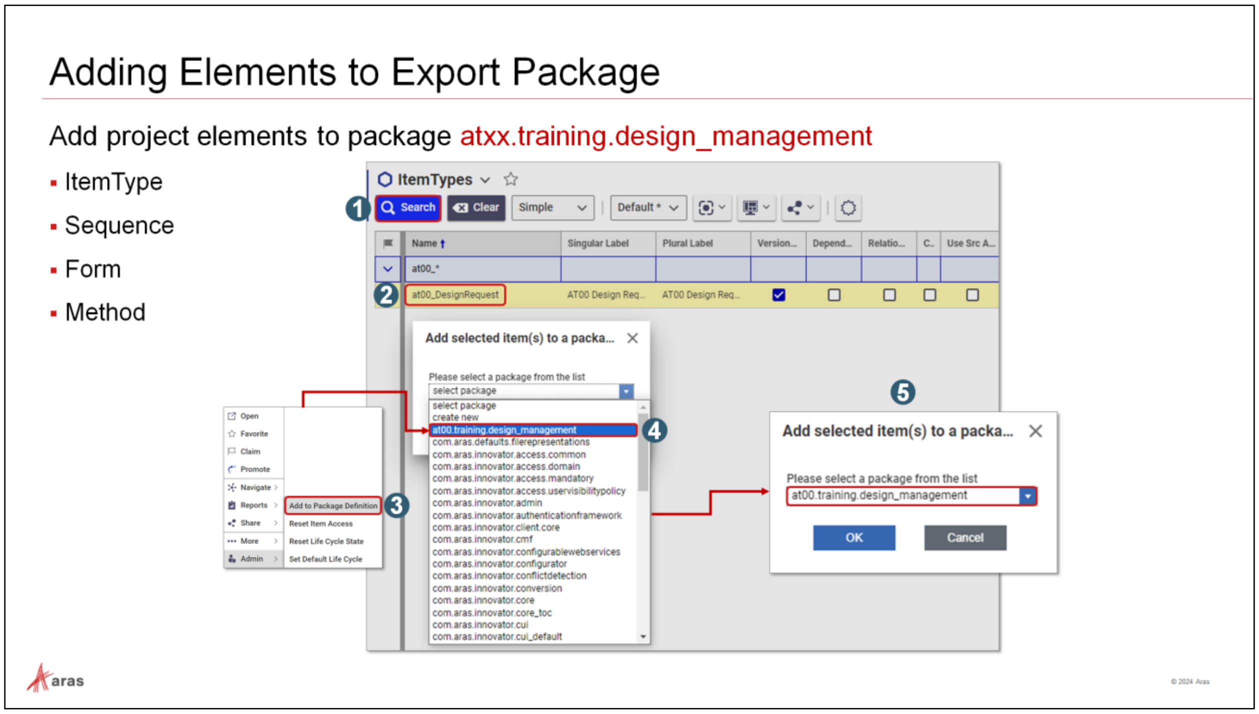
© 2024 Aras

Subscribing Method to Server Events

Follow the steps below to subscribe the newly method to the **OnBeforeAdd** and **OnBeforeUpdate** to ensure newly created or existing **Design Requests** cannot be saved without a **Completion Date** if the **Patent?** checkbox is selected.

Try It … Subscribe Method to Server Events

1. Navigate to **Administration** > **ItemTypes** and open the *atxx_DesignRequest* ItemType in **Edit** mode.
2. Assign the *atxx_DesignRequestValidate* method to the **OnBeforeAdd** and **OnBeforeUpdate** **Server Events**.
3. Click the **Done** button to save changes to the new Design Request ItemType.



Adding Elements to Export Package

Add project elements to package atxx.training.design_management

- ItemType
- Sequence
- Form
- Method

ItemTypes
Search
Clear
Simple
Default *
Name
Singular Label
Plural Label
Version...
Depend...
Relatio...
C...
Use Src A...
at00_*
at00_DesignRequest
AT00 Design Req...
AT00 Design Req...
Open
Favorite
Claim
Promote
Navigate
Reports
Share
More
Admin
Add to Package Definition
Reset Item Access
Reset Life Cycle State
Set Default Life Cycle
Add selected item(s) to a packa...
Please select a package from the list
select package
select package
create new
at00.training.design_management
com.aras.defaults.filerepresentations
com.aras.innovator.access.common
com.aras.innovator.access.domain
com.aras.innovator.access.mandatory
com.aras.innovator.access.uservisibilitypolicy
com.aras.innovator.admin
com.aras.innovator.authenticationframework
com.aras.innovator.client.core
com.aras.innovator.cmf
com.aras.innovator.configurablewebservices
com.aras.innovator.configurator
com.aras.innovator.conflictdetection
com.aras.innovator.conversion
com.aras.innovator.core
com.aras.innovator.core_toc
com.aras.innovator.cui
com.aras.innovator.cui_default
Add selected item(s) to a packa...
Please select a package from the list
at00.training.design_management
OK
Cancel
1
2
3
4
5
aras
© 2024 Aras

Adding Elements to Export Package

Follow the steps below to add the following elements to the *atxx.training.design_management* export package:

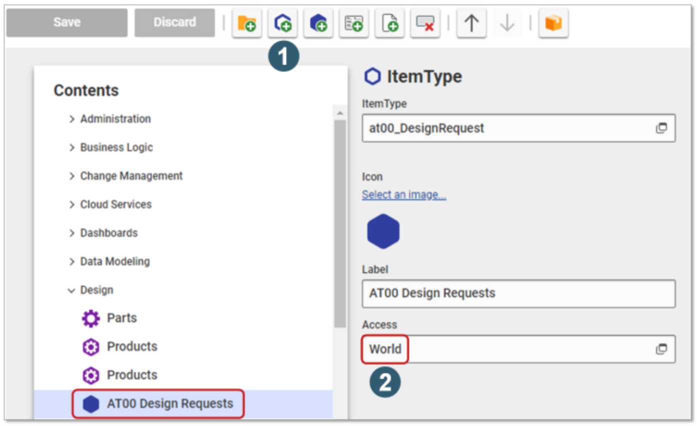
- ItemType
- Sequence
- Form
- Method

Try it … Add Elements to Export Package

1. In TOC go to menu **Administration > ItemTypes**, and at its right click on **Search** button.
2. Use any search criteria to find your *atxx_DesignRequest* **ItemType**.
3. Right-click on your **ItemType** and select menu **Admin > Add to Package Definition** from context menus.
4. Select the *atxx.training.design_management* export package and click **OK** to confirm choice.
5. Repeat above steps for all elements to be exported.
6. In the *atxx.training.design_management* export package, go to each package group to verify its contents.

Adding New ItemType to TOC

- Open TOC Editor in Administration menu
- Add atxx_DesignRequest ItemType to Design category
- Ensure World Identity can access



ItemType: at00_DesignRequest
Icon
Label: AT00 Design Requests
Access: World

aras © 2024 Aras

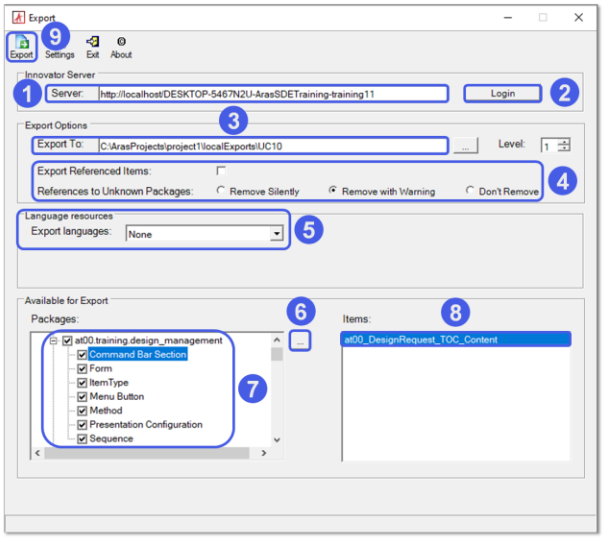
Adding New ItemType to TOC

Follow the steps below to add the new *atxx_DesignRequest* ItemType to the root of the TOC **Design** category.

Try it … Add New ItemType to TOC

1. In TOC go to menu **Administration** > **Configuration** > **TOC Editor**.
2. Select the **Design** category and click on **Add ItemType** button.
3. Find your *atxx_DesignRequest* **ItemType** and ensure **World** is selected in the **Access** field.
4. Click the **Save** button when done.
5. Create a few Design Requests and test to make sure the method logic works correctly:
    a. An error is returned if one tries to save a new Design Request with a **Patent** but without a **Completion Date**.
    b. An error is returned if one tries to update and save an existing Design Request with a **Patent** but without a **Completion Date**.
6. In TOC go to menu **Administration > Configuration > PackageDefinitions**.
7. Find your *atxx.training.design_management* export package and ensure it contains the following additional package groups: **Presentation Configuration**, **Command Bar Section**, and **Menu Button**.

Exporting Contents of New Package Definition



Innovator Server
Server: http://localhost/DESKTOP-5467N2U-ArasSDETraining-training11
Login

Export Options
Export To: C:\ArasProjects\project1\localExports\UC10
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:
- ☑ at00.training.design_management
  - ☑ Command Bar Section
  - ☑ Form
  - ☑ ItemType
  - ☑ Menu Button
  - ☑ Method
  - ☑ Presentation Configuration
  - ☑ Sequence

Items:
at00_DesignRequest_TOC_Content

1. Provide the Server URL
2. Select DB and provide credentials in Login dialog
3. Provide the destination for export
   - *C:\ArasProjects\project1\localExports\UC10*
4. Set additional Export and Reference Options
5. Set the Language resources (if required)
6. Refresh the list of packages
7. Expand and select the desired package and groups to export
8. Select the elements to export
9. Start Export

Exporting Contents of New Package Definition

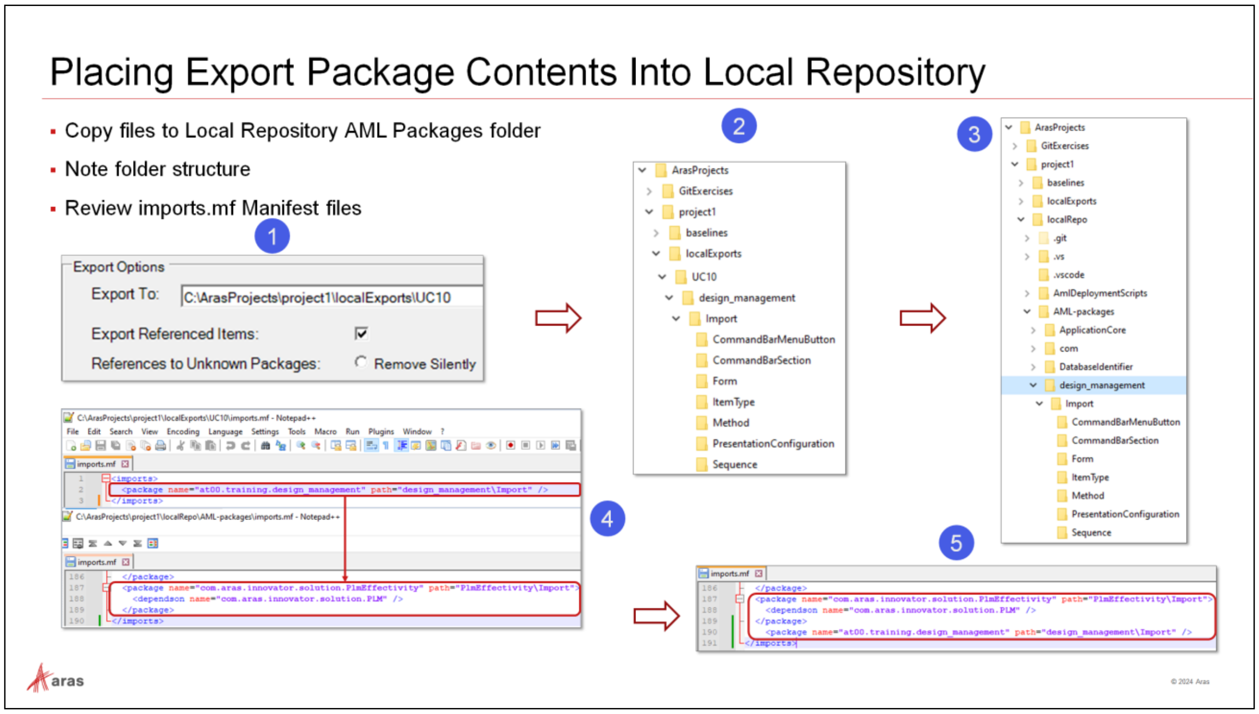
The Package Definition now needs to be exported. It is recommended that you export the package to a temporary location (e.g., localExports directory) and review the **imports.mf** manifest file to ensure the appropriate dependencies have been defined. The local export files in the staging area must then be copied/merged into the **AML-packages** folder in the **trainingxx** branch of the repository.

Try It … Export Contents of New Package Definition

1. Start the Aras Export tool and login to the instance of Aras Innovator with the admin credentials.
2. Set the Export folder to a staging folder on the developer machine.
   Example `C:\ArasProjects\Project1\LocalExport\UC10`
3. Select the Package elements and export to the local folder.
4. Examine the localExports folder contents to ensure the appropriate folders and AML files have been created/changed.

Notes:

- Remember that the **imports.mf** manifest file is updated each time an export is executed in the localExports directory. You should check this file to make sure the appropriate dependencies have been configured if necessary.
- Because there is only one **imports.mf** file for the project in the repository AML-packages folder be careful not to copy over changes from a previous configuration.



Placing Export Package Contents Into Local Repository

- Copy files to Local Repository AML Packages folder
- Note folder structure
- Review imports.mf Manifest files

Export Options
Export To: C:\ArasProjects\project1\localExports\UC10
Export Referenced Items: ✓
References to Unknown Packages: Remove Silently

C:\ArasProjects\project1\localExports\UC10\imports.mf - Notepad++

```
<imports>
  <package name="at00.training.design_management" path="design_management\Import" />
</imports>
```

C:\ArasProjects\project1\localRepo\AML-packages\imports.mf - Notepad++

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

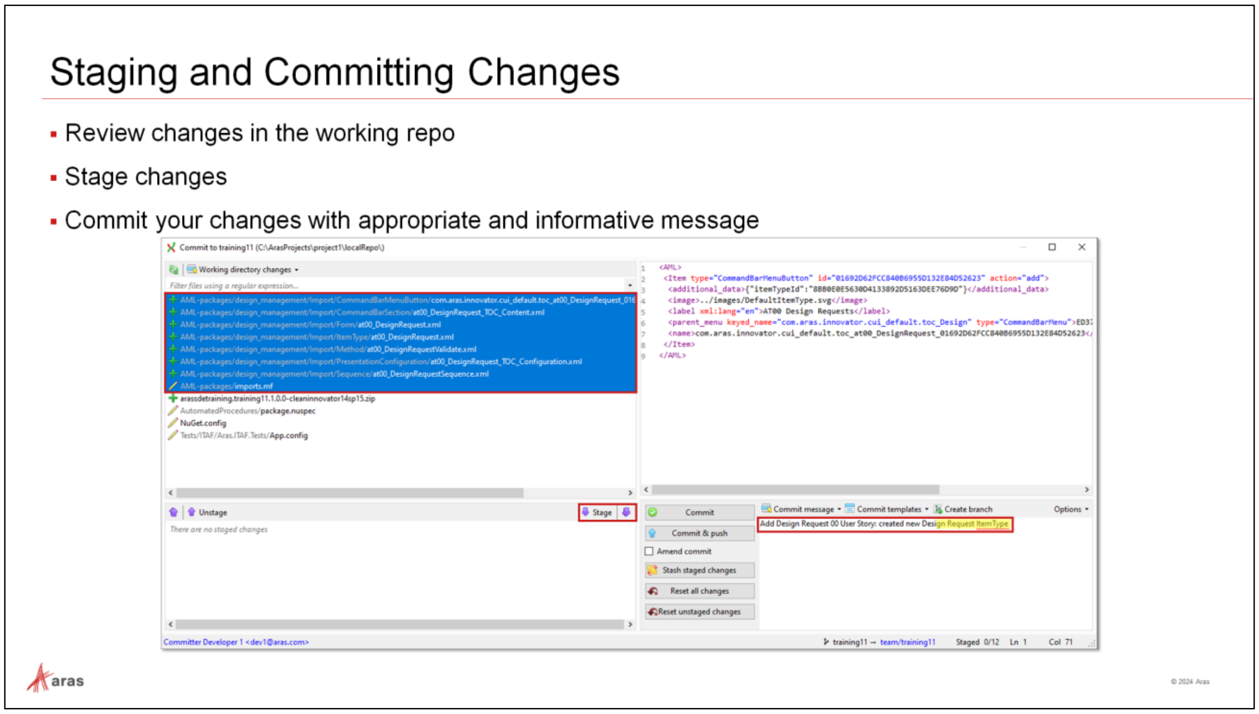
```
  </package>
  <package name="com.aras.innovator.solution.PlmEffectivity" path="PlmEffectivity\Import">
    <dependson name="com.aras.innovator.solution.PLM" />
  </package>
    <package name="at00.training.design_management" path="design_management\Import" />
</imports>
```

Placing Export Package Contents Into Local Repository

It is often necessary, as in this case, to export to a local folder then copy what is necessary into the local repository.

Try It … Place Export Package Contents Into Local Repository

1. Copy and paste the folder **C:\ArasProjects\project1\localExports\UC10\design_management** into **C:\ArasProjects\Project1\LocalRepo\AML-packages**.
2. Accept prompts for replacement if encountered.
3. Copy the contents of the newly created **imports.mf** file to the existing **imports.mf** file in the **AML-packages** folder.
   Ensure that the existing **imports.mf** file includes the following line:
   ```
   <package name="at00.training.design_management" path="design_management\Import" />
   ```



Staging and Committing Changes

- Review changes in the working repo
- Stage changes
- Commit your changes with appropriate and informative message

Commit to training11 (C:\ArasProjects\project1\localRepo\)

Working directory changes

```
AML-packages/design_management/Import/CommandBarMenuButton/com.aras.innovator.cui_default.toc_at00_DesignRequest_016
AML-packages/design_management/Import/CommandBarSection/at00_DesignRequest_TOC_Content.xml
AML-packages/design_management/Import/Form/at00_DesignRequest.xml
AML-packages/design_management/Import/ItemType/at00_DesignRequest.xml
AML-packages/design_management/Import/Method/at00_DesignRequestValidate.xml
AML-packages/design_management/Import/PresentationConfiguration/at00_DesignRequest_TOC_Configuration.xml
AML-packages/design_management/Import/Sequence/at00_DesignRequestSequence.xml
AML-packages/imports.mf
arassdetraining.training11.1.0.0-cleaninnovator14sp15.zip
AutomatedProcedures/package.nuspec
NuGet.config
Tests/ITAF/Aras.ITAF.Tests/App.config
```

```
<AML>
 <Item type="CommandBarMenuButton" id="01692D62FCC840B6955D132E84D52623" action="add">
  <additional_data>{"itemTypeId":"8BB0E0E5630D4133892D5163DEE76D9D"}</additional_data>
  <image>../images/DefaultItemType.svg</image>
  <label xml:lang="en">AT00 Design Requests</label>
  <parent_menu keyed_name="com.aras.innovator.cui_default.toc_Design" type="CommandBarMenu">ED3
  <name>com.aras.innovator.cui_default.toc_at00_DesignRequest_01692D62FCC840B6955D132E84D52623</
 </Item>
</AML>
```

There are no staged changes

Commit message: Add Design Request 00 User Story: created new Design Request ItemType

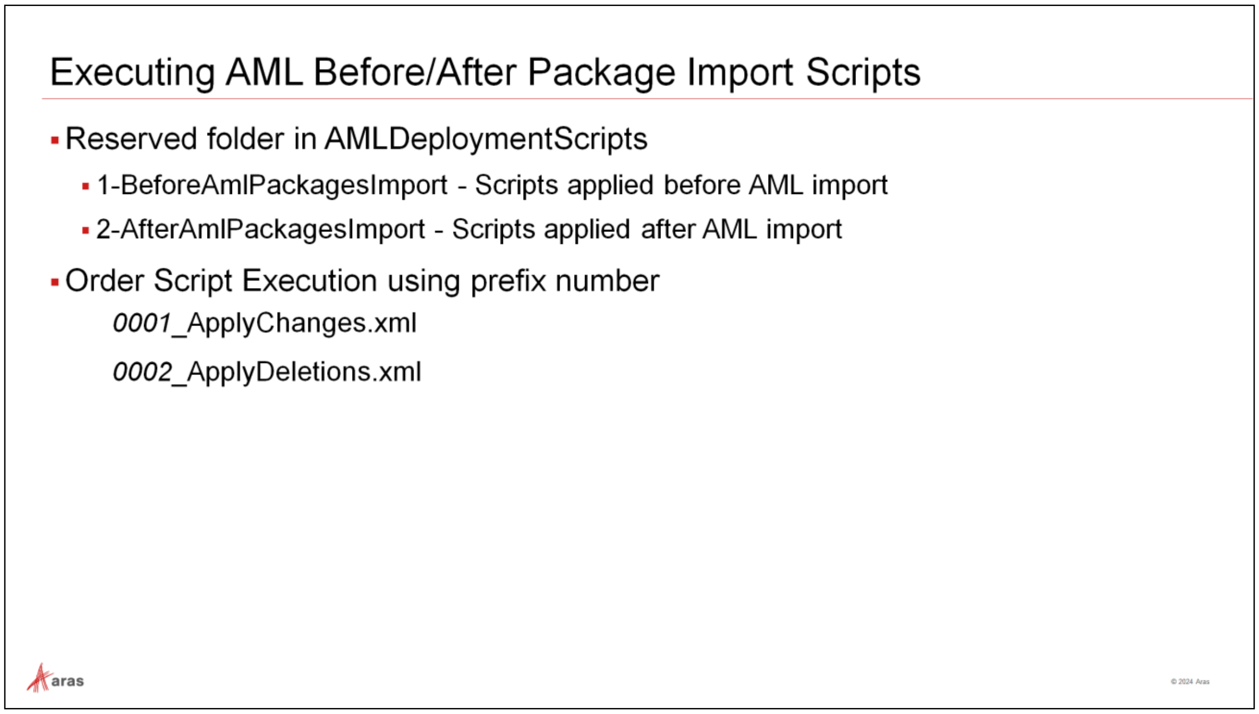
Committer Developer 1 <dev1@aras.com>   training11 → team/training11   Staged 0/12   Ln 1   Col 71

Staging and Committing Changes

It is preferable to do atomic commits to better control your work; for this reason, you may want to split your work, by first committing all configuration work, and then committing all code-related work in other commits, in case you need to fix some of the code in Visual Studio.

Try It … Stage and Commit Changes

1. Navigate to **C:\ArasProjects\project1\LocalRepo\AML-packages\design_management\Import** and note its contents.
2. Open Git Bash and Git Extensions for the repository **..\project1\LocalRepo**.
3. Verify the status of the repository in Git Bash by running *git status*.
4. In Git Extensions, go to the **Commit to trainingxx** dialog, and stage the different files, i.e., **atxx_DesignRequest.xml**, **atxx_DesignRequest.xml**, **imports.mf**, etc. (7 files in total).
5. Verify now the new status of the repository in Git Bash by running *git status*.
6. In Git Extensions, commit the xml files above in the **Commit to trainingxx** dialog, after addition of a proper commit message, e.g., "*Add Design Request 00 User Story: created new Design Request ItemType*".
7. Click **OK** in **Process** dialog.
8. Verify again the new status of the repository in Git Bash by running *git status*.



Executing AML Before/After Package Import Scripts

- Reserved folder in AMLDeploymentScripts
  - 1-BeforeAmlPackagesImport - Scripts applied before AML import
  - 2-AfterAmlPackagesImport - Scripts applied after AML import
- Order Script Execution using prefix number

  *0001*_ApplyChanges.xml

  *0002*_ApplyDeletions.xml

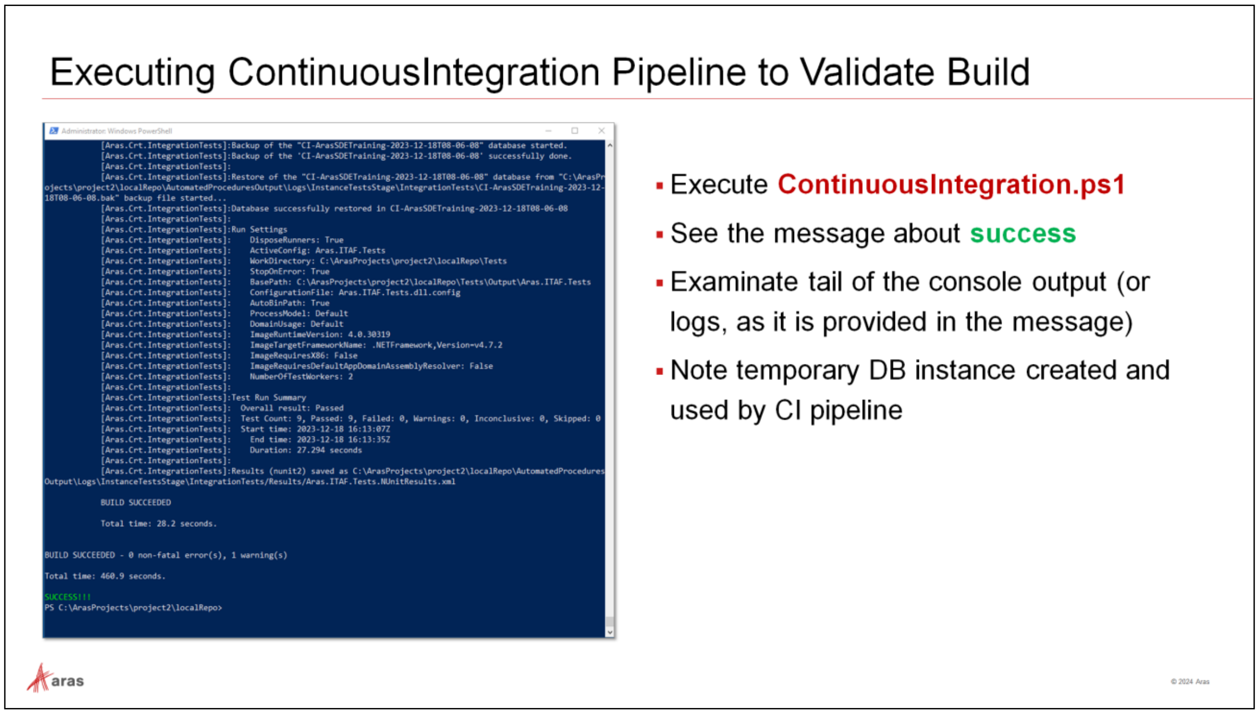
Executing AML Before/After Package Import Scripts

If database modifications need to be made before or after importing the AML-packages, two reserved folders in the **AMLDeploymentScripts** folder can be used to supply additional AML statements.

Some use cases for creating custom AML may be:

- Modify or remove metadata that prevents a successful import of AML packages (for example: server event constraints)
- Resolve circular dependencies in the AML-packages by correcting the Packages configuration in the DB
- Add or modify any data (not metadata) that is not in the AML packages

Scripts are executed in alphabetic order, so it is recommended to prefix the XML file with a number which indicates scripts execution order.



Executing ContinuousIntegration Pipeline to Validate Build

```
           [Aras.Crt.IntegrationTests]:Backup of the "CI-ArasSDETraining-2023-12-18T08-06-08" database started.
           [Aras.Crt.IntegrationTests]:Backup of the 'CI-ArasSDETraining-2023-12-18T08-06-08' successfully done.
           [Aras.Crt.IntegrationTests]:
           [Aras.Crt.IntegrationTests]:Restore of the "CI-ArasSDETraining-2023-12-18T08-06-08" database from "C:\ArasPr
ojects\project2\localRepo\AutomatedProceduresOutput\Logs\InstanceTestsStage\IntegrationTests\CI-ArasSDETraining-2023-12-
18T08-06-08.bak" backup file started...
           [Aras.Crt.IntegrationTests]:Database successfully restored in CI-ArasSDETraining-2023-12-18T08-06-08
           [Aras.Crt.IntegrationTests]:
           [Aras.Crt.IntegrationTests]:Run Settings
           [Aras.Crt.IntegrationTests]:    DisposeRunners: True
           [Aras.Crt.IntegrationTests]:    ActiveConfig: Aras.ITAF.Tests
           [Aras.Crt.IntegrationTests]:    WorkDirectory: C:\ArasProjects\project2\localRepo\Tests
           [Aras.Crt.IntegrationTests]:    StopOnError: True
           [Aras.Crt.IntegrationTests]:    BasePath: C:\ArasProjects\project2\localRepo\Tests\Output\Aras.ITAF.Tests
           [Aras.Crt.IntegrationTests]:    ConfigurationFile: Aras.ITAF.Tests.dll.config
           [Aras.Crt.IntegrationTests]:    AutoBinPath: True
           [Aras.Crt.IntegrationTests]:    ProcessModel: Default
           [Aras.Crt.IntegrationTests]:    DomainUsage: Default
           [Aras.Crt.IntegrationTests]:    ImageRuntimeVersion: 4.0.30319
           [Aras.Crt.IntegrationTests]:    ImageTargetFrameworkName: .NETFramework,Version=v4.7.2
           [Aras.Crt.IntegrationTests]:    ImageRequiresX86: False
           [Aras.Crt.IntegrationTests]:    ImageRequiresDefaultAppDomainAssemblyResolver: False
           [Aras.Crt.IntegrationTests]:    NumberOfTestWorkers: 2
           [Aras.Crt.IntegrationTests]:
           [Aras.Crt.IntegrationTests]:Test Run Summary
           [Aras.Crt.IntegrationTests]:  Overall result: Passed
           [Aras.Crt.IntegrationTests]:  Test Count: 9, Passed: 9, Failed: 0, Warnings: 0, Inconclusive: 0, Skipped: 0
           [Aras.Crt.IntegrationTests]:  Start time: 2023-12-18 16:13:07Z
           [Aras.Crt.IntegrationTests]:    End time: 2023-12-18 16:13:35Z
           [Aras.Crt.IntegrationTests]:    Duration: 27.294 seconds
           [Aras.Crt.IntegrationTests]:
           [Aras.Crt.IntegrationTests]:Results (nunit2) saved as C:\ArasProjects\project2\localRepo\AutomatedProcedures
Output\Logs\InstanceTestsStage\IntegrationTests/Results/Aras.ITAF.Tests.NUnitResults.xml

           BUILD SUCCEEDED

           Total time: 28.2 seconds.


BUILD SUCCEEDED - 0 non-fatal error(s), 1 warning(s)

Total time: 460.9 seconds.

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

- Execute **ContinuousIntegration.ps1**
- See the message about **success**
- Examinate tail of the console output (or logs, as it is provided in the message)
- Note temporary DB instance created and used by CI pipeline

Executing ContinuousIntegration Pipeline to Validate Build

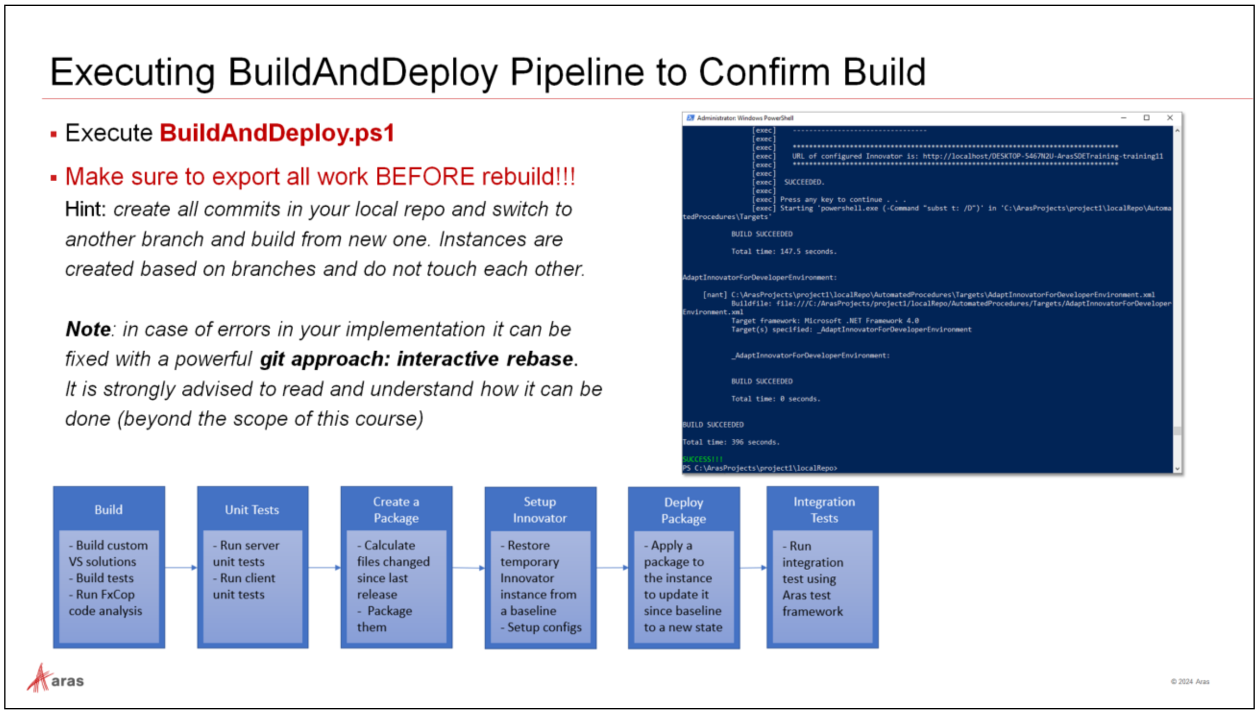
Validate the Build by executing the ContinuousIntegration.ps1 pipeline.

Pay attention to the results: the build scripts run an additional set of checks that treat warnings as errors. As a result, you may see errors when running the ContinuousIntegration pipeline.

It is recommended to run the CI pipeline whenever you create or update methods: it is in general a good DevOps CI practice to do so as it is fast and helps reduce issues in your project.

Try It … Execute ContinuousIntegration Pipeline to Validate Build

1. Open an admin PowerShell session and navigate to the local repository at **C:\ArasProjects\project1\LocalRepo**.
2. Run script: *.\ContinuousIntegration.ps1*.
3. Examine the output of the pipeline, and fix issues in case of failure.



Executing BuildAndDeploy Pipeline to Confirm Build

- Execute **BuildAndDeploy.ps1**
- Make sure to export all work BEFORE rebuild!!!

  Hint: *create all commits in your local repo and switch to another branch and build from new one. Instances are created based on branches and do not touch each other.*

  ***Note**: in case of errors in your implementation it can be fixed with a powerful **git approach: interactive rebase**.*
  It is strongly advised to read and understand how it can be done (beyond the scope of this course)

```
Administrator: Windows PowerShell
                [exec]    --------------------------------
                [exec]
                [exec]    ******************************************************************************
                [exec]    URL of configured Innovator is: http://localhost/DESKTOP-5467N2U-ArasSDETraining-training11
                [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: 147.5 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: 396 seconds.

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

Build
- Build custom VS solutions
- Build tests
- Run FxCop code analysis
Unit Tests
- Run server unit tests
- Run client unit tests
Create a Package
- Calculate files changed since last release
- Package them
Setup Innovator
- Restore temporary Innovator instance from a baseline
- Setup configs
Deploy Package
- Apply a package to the instance to update it since baseline to a new state
Integration Tests
- Run integration test using Aras test framework
aras
© 2024 Aras

Executing BuildAndDeploy Pipeline to Confirm Build

Once you have added and committed the project files in the AML Packages directory, you can rebuild the environment at any time using the **BuildAndDeploy.ps1** pipeline. Aras Innovator and the original (baseline) database will be reinstalled, and any AML Packages will be applied after installation.

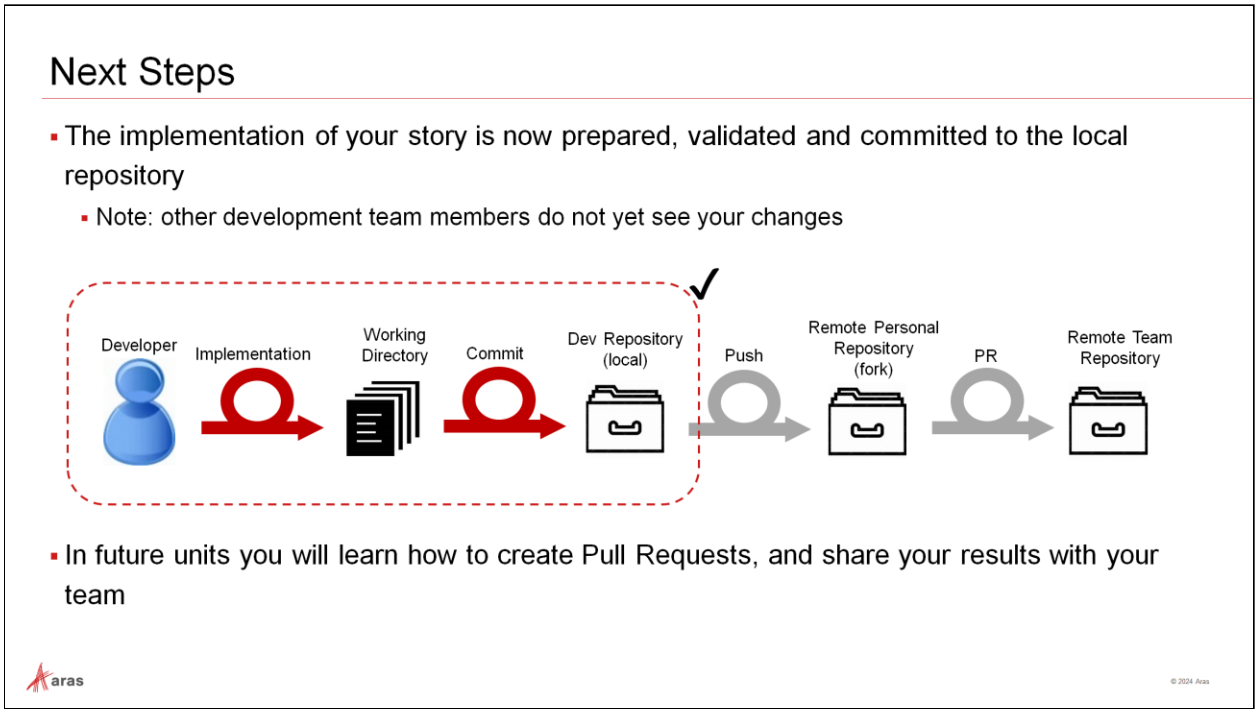
Make sure that the export files have been successfully created and stored in the AML Packages folder in the development repository.

Try It … Execute BuildAndDeploy Pipeline to Confirm Build

1. Access the repository directory and make sure the current sprint branch (**trainingxx**) has been checked out.
2. Open an admin PowerShell session and navigate to the local repository at **C:\ArasProjects\project1\LocalRepo**.
3. Run script: *.\BuildAndDeploy.ps1*.
4. If the build is successful, a green prompt will appear. If the build fails, examine the log entries to determine the problem (e.g., missing AML files?).

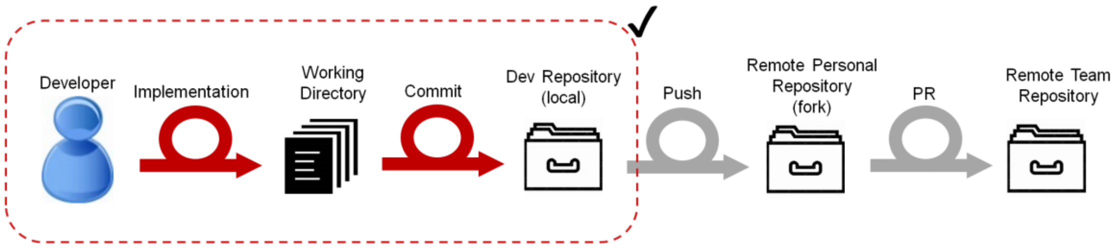
Notes:

- To avoid the issue of losing changes you have not previously exported you should execute the **ContinuousIntegration.ps1** pipeline first.
- You may also run **BuildAndDeploy** first in a different branch to carry out some tests and avoid wiping out all your changes from your LDE Innovator instance.



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



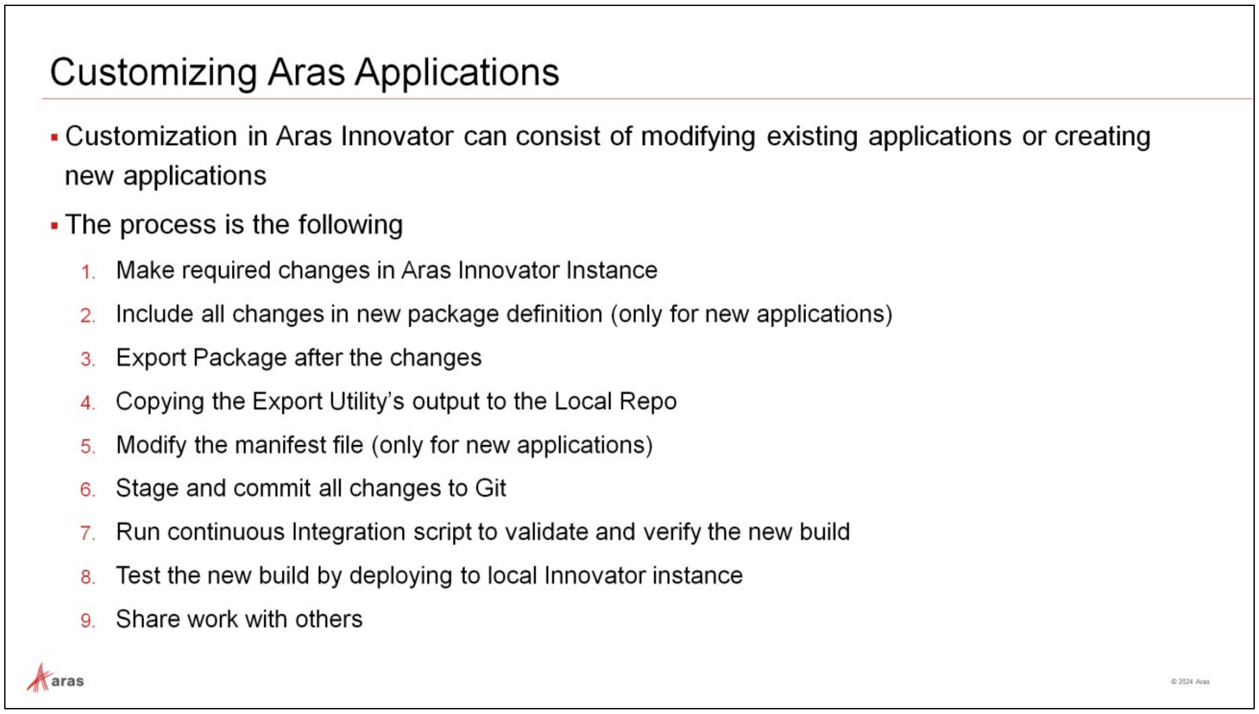
- In future units you will learn how to create Pull Requests, and share your results with your team

Next Steps

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

The next few steps will involve the following activities:

- Fetching and rebasing from/onto the Team's repository **trainingxx** branch to ensure synchronization, and minimize issues
- Pushing our locally-committed work to our personal fork **trainingxx** branch
- Creating a pull request to propose our work to the rest of the team for inclusion in the Team's repository **trainingxx** branch



Customizing Aras Applications

- Customization in Aras Innovator can consist of modifying existing applications or creating new applications
- The process is the following
  1. Make required changes in Aras Innovator Instance
  2. Include all changes in new package definition (only for new applications)
  3. Export Package after the changes
  4. Copying the Export Utility's output to the Local Repo
  5. Modify the manifest file (only for new applications)
  6. Stage and commit all changes to Git
  7. Run continuous Integration script to validate and verify the new build
  8. Test the new build by deploying to local Innovator instance
  9. Share work with others

aras

© 2024 Aras

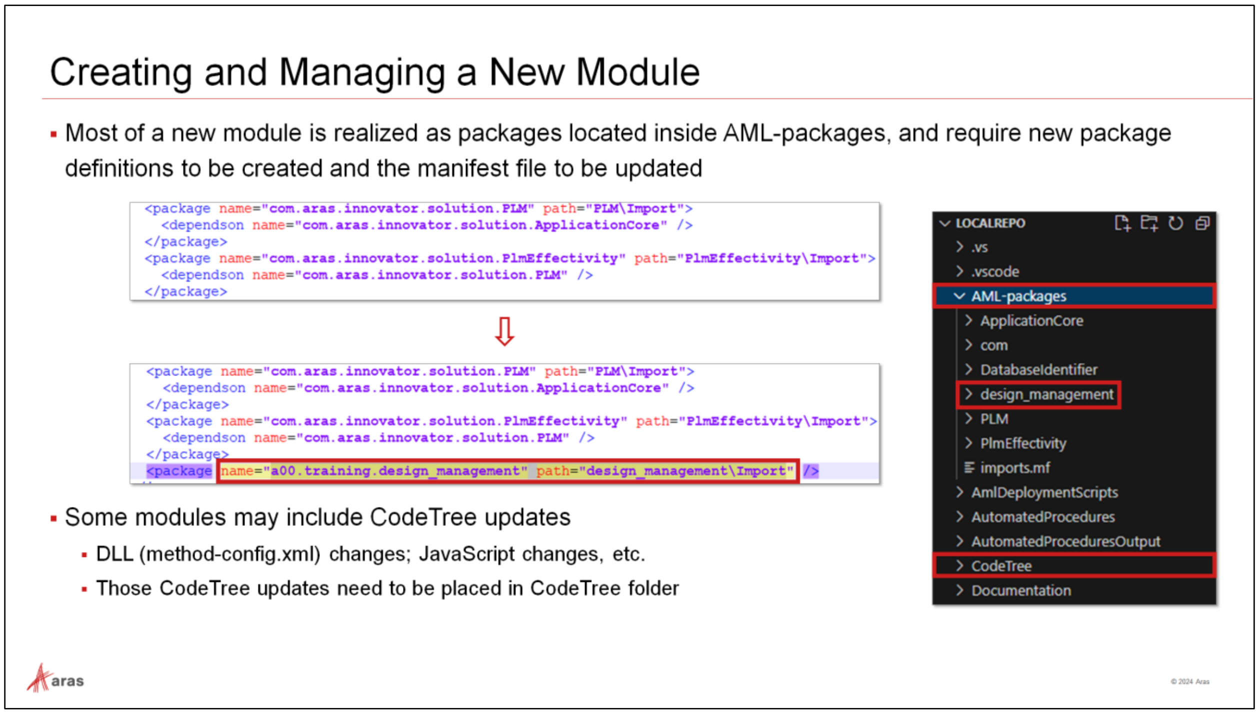
Customizing Aras Applications

Customization in Aras Innovator can consist of modifying existing applications or creating new applications.

The process is the following:

1. Make required changes in Aras Innovator Instance.
   Login into the Innovator Instance and make customizations as per the project requirements.
2. Include all changes in new package definition (only for new applications).
   Once all changes are complete, include them in the Package Definition.
3. Export Package after the changes.
   Open the Export Utility and add the necessary details to prepare for export.
4. Copying the Export Utility's output to the Local Repo.
   Once the changes are made and the files are exported, copy the top common folder, and paste it into the local working directory of the repository (AML-Packages). During this process, be sure to accept any file replacement warnings that may appear.
5. Modify the manifest file (only for new applications)
   Ensure the manifest file includes any new packages

   Stage and commit all changes to Git.
   Once the necessary changes have been made to the files, it is important to stage and commit them to the version control system (Git).
6. Run continuous Integration script to validate and verify the new build.
7. Test the new build by deploying it to local Innovator instance.
8. Share work with others.



Creating and Managing a New Module

- Most of a new module is realized as packages located inside AML-packages, and require new package definitions to be created and the manifest file to be updated

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

⇩

```
<package name="com.aras.innovator.solution.PLM" path="PLM\Import">
  <dependson name="com.aras.innovator.solution.ApplicationCore" />
</package>
<package name="com.aras.innovator.solution.PlmEffectivity" path="PlmEffectivity\Import">
  <dependson name="com.aras.innovator.solution.PLM" />
</package>
<package name="a00.training.design_management" path="design_management\Import" />
```

```
LOCALREPO
  > .vs
  > .vscode
  v AML-packages
    > ApplicationCore
    > com
    > DatabaseIdentifier
    > design_management
    > PLM
    > PlmEffectivity
      imports.mf
  > AmlDeploymentScripts
  > AutomatedProcedures
  > AutomatedProceduresOutput
  > CodeTree
  > Documentation
```

- Some modules may include CodeTree updates
  - DLL (method-config.xml) changes; JavaScript changes, etc.
  - Those CodeTree updates need to be placed in CodeTree folder

aras

© 2024 Aras

Creating and Managing a New Module

A new module normally results in a new AML package with its own package definition. The package definition contains all changes made to instances of Aras Innovator.

Several steps are needed for a new package in SDE:

- Definition of the package collecting all the needed items and specifying dependencies from other packages
- Export of the package to the file system
- Locate the results properly in the repository directories
- Adapt the manifest file to contain the previous packages, but also the new one
- Assign the new and modified files to the Git repository with commit or stage including the manifest file

Following those steps will ensure the package becomes part of the next build.

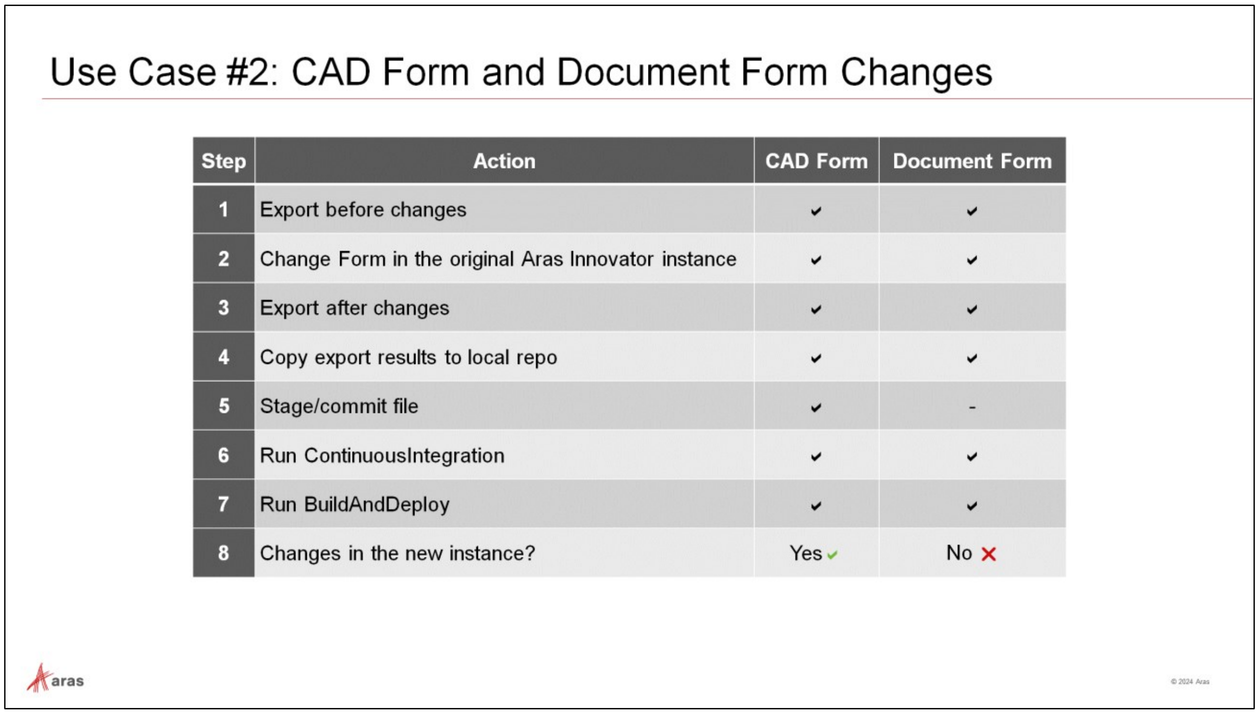
If you create a new package, it is recommended to use Java naming conventions for packages (lowercase and dots indicating a hierarchy) and align folder names with the packages they contain.

In some use cases, modules require additional changes directly to the code tree. Such changes do not need additional packaging, but they also need to be positioned in the correct folder and be assigned to the Git repository.

Examples for such exceptions:

- Adding images for own icons in the customer folder
- Adding a DLL for federation and assigning it to the file method-config.xml

Note: Whenever you have updated any sections of the repository, the new or modified files need to be staged and committed to become part of the next build.



Use Case #2: 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 #2: CAD Form and Document Form Changes

The idea behind the following Use Case #2 (UC2) 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 take 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 changes to Git.

Since you have added the properties to existing items, there is no need to update the import manifest file in this UC2, 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.

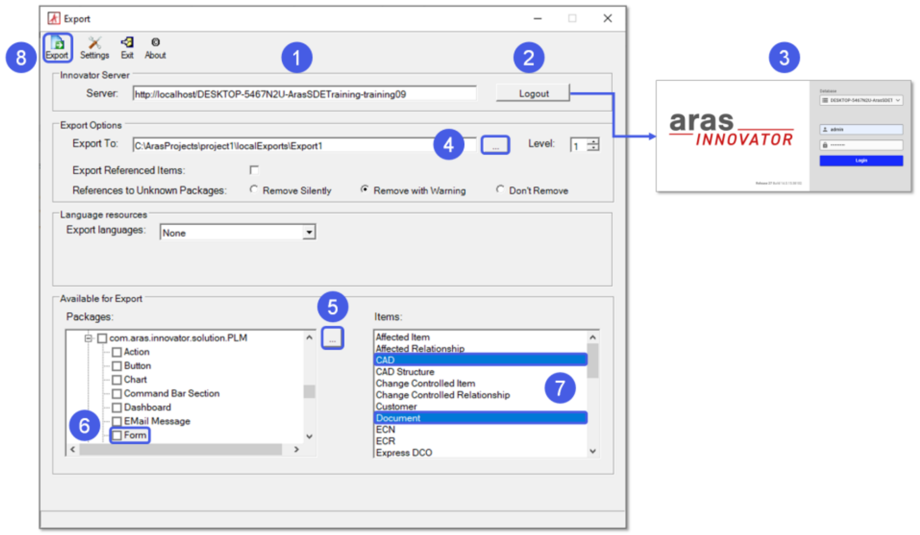
The UC2 steps include checks and verifications which help to understand the details of the process as listed in the table.

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



Innovator Server

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

Export Options

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

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 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.

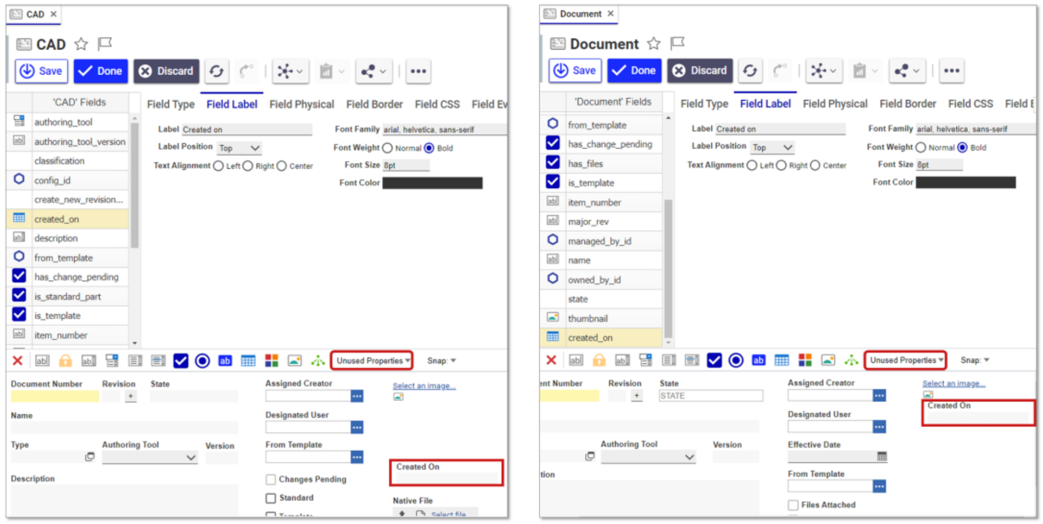
Note: You will be asked if you want to create a new directory or if you want to overwrite, respectively. Simply accept the option to create the new directory.

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
Font Family: arial, helvetica, sans-serif
Font Weight: Bold
Font Size: 8pt

aras

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

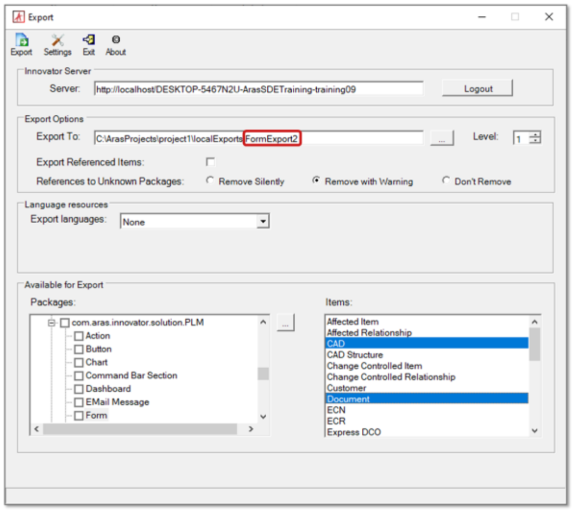
1. Login to Aras Innovator as user admin.
2. Select **Administration > Forms** and search for the **CAD** Form.
3. Switch to **Edit** mode.
4. Click on **Unused Properties** and select **created_on**.
5. Place the property on the form, change the label to **Created on** and save the form.

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



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

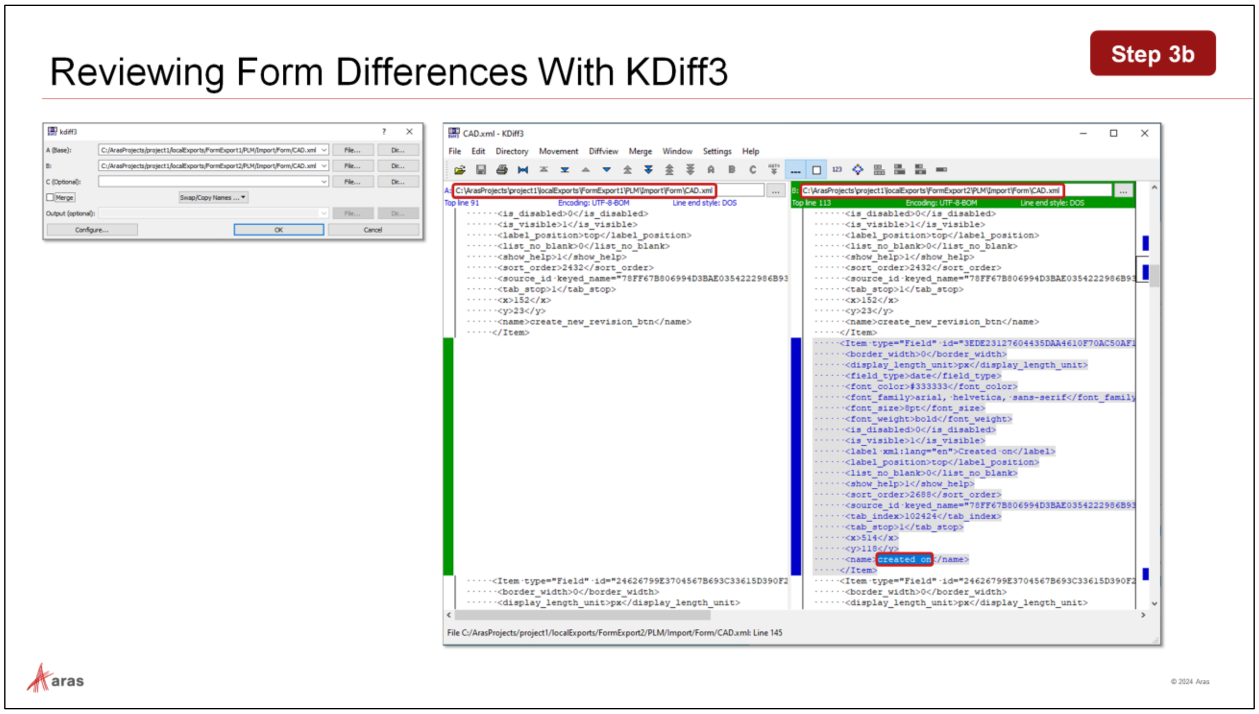
aras

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

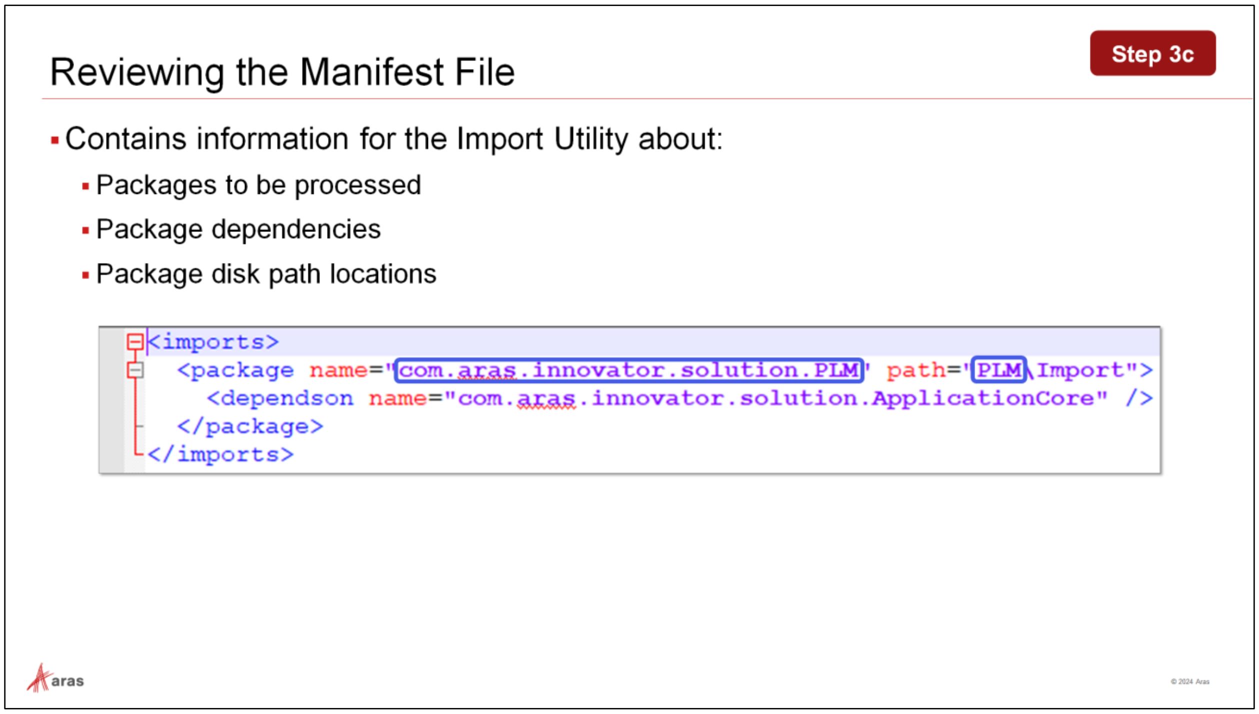
As a good exercise, analyze the differences between the export data before and after changes. We want to avoid situations where additional changes (currently not relevant) are present before we make a commit.

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>
```

aras
© 2024 Aras

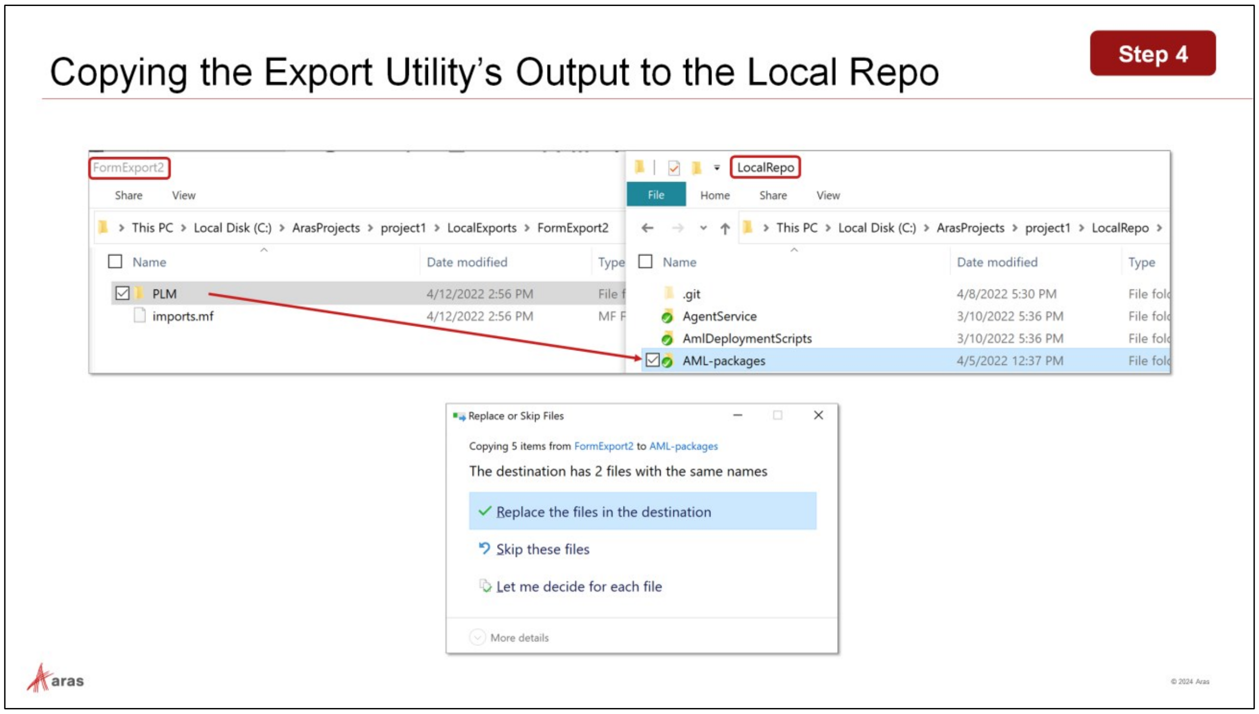
Reviewing the Manifest File

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 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

> This PC > Local Disk (C:) > ArasProjects > project1 > LocalExports > FormExport2

Name	Date modified	Type
PLM	4/12/2022 2:56 PM	File f
imports.mf	4/12/2022 2:56 PM	MF F

LocalRepo

> This PC > Local Disk (C:) > ArasProjects > project1 > LocalRepo >

Name	Date modified	Type
.git	4/8/2022 5:30 PM	File fol
AgentService	3/10/2022 5:36 PM	File fol
AmlDeploymentScripts	3/10/2022 5:36 PM	File fol
AML-packages	4/5/2022 12:37 PM	File fol

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

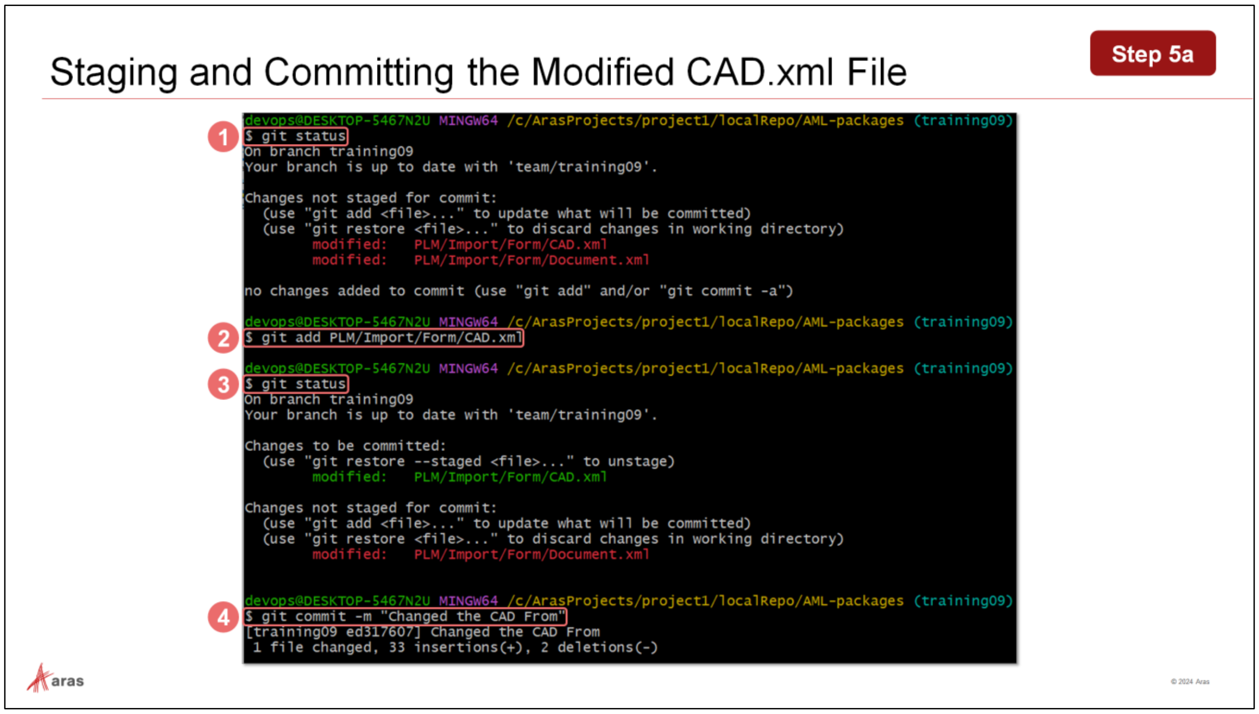
More details

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

```
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

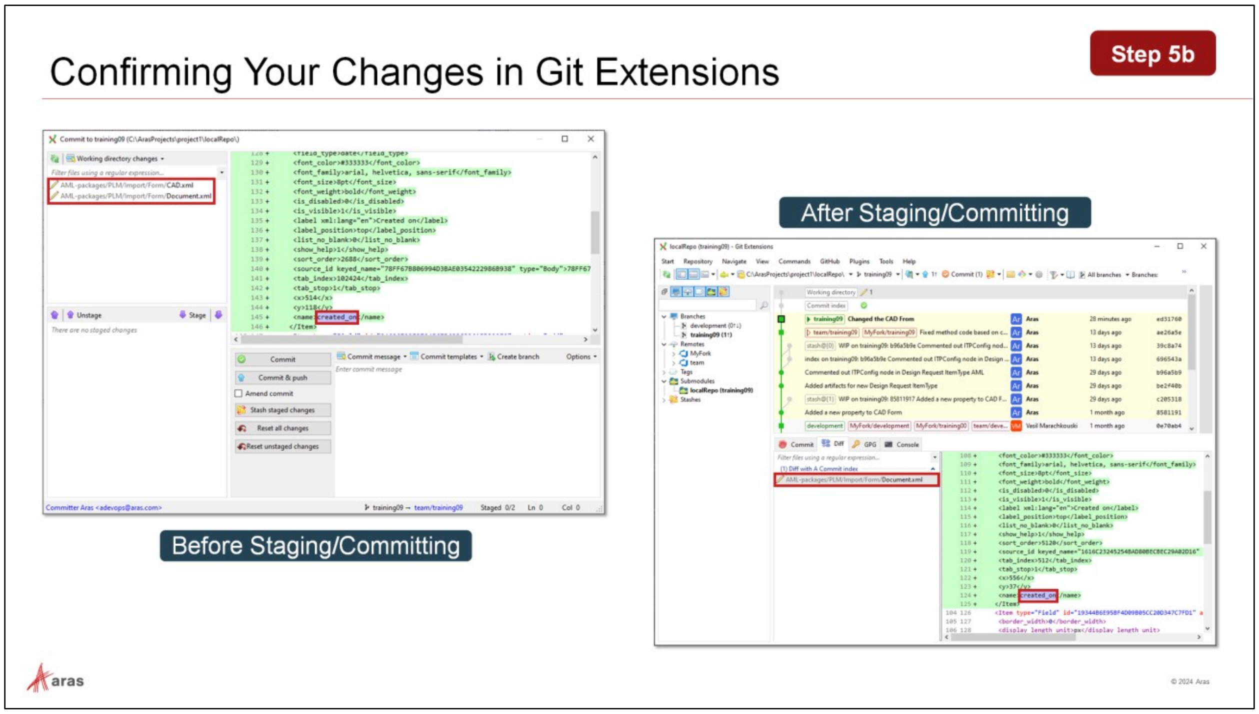
To add the modified file CAD.xml to the repository, we stage it by using the commands *git add …/CAD.xml* and *commit - m "commit message"*.

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

Before Staging/Committing

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

Working directory changes

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

Unstage | Stage

There are no staged changes

```
<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="78FF67B886994D3BAE0354222986B938" type="Body">78FF67
<tab_index>102424</tab_index>
<tab_stop>1</tab_stop>
<x>514</x>
<y>118</y>
<name>created_on</name>
</Item>
```

Commit | Commit & push | Amend commit | Stash staged changes | Reset all changes | Reset unstaged changes

Commit message | Commit templates | Create branch | Options

Enter commit message

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

After Staging/Committing

localRepo (training09) - Git Extensions

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

Commit | Diff | GPG | Console

(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="19344B6E95BF4D09B05CC20D347C7FD1"
<border_width>0</border_width>
<display_length_unit>px</display_length_unit>
```

aras

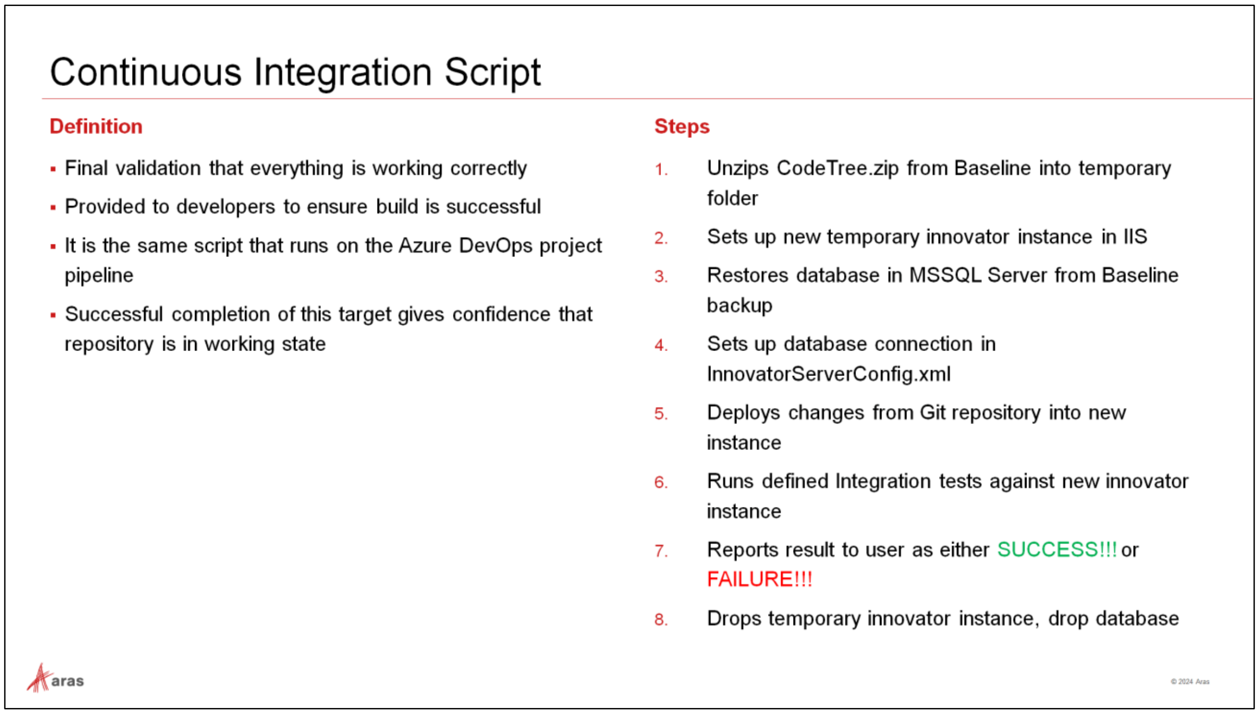
© 2024 Aras

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).



Continuous Integration Script

Definition

- Final validation that everything is working correctly
- Provided to developers to ensure build is successful
- It is the same script that runs on the Azure DevOps project pipeline
- Successful completion of this target gives confidence that repository is in working state

Steps

1. Unzips CodeTree.zip from Baseline into temporary folder
2. Sets up new temporary innovator instance in IIS
3. Restores database in MSSQL Server from Baseline backup
4. Sets up database connection in InnovatorServerConfig.xml
5. Deploys changes from Git repository into new instance
6. Runs defined Integration tests against new innovator instance
7. Reports result to user as either SUCCESS!!! or FAILURE!!!
8. Drops temporary innovator instance, drop database

aras

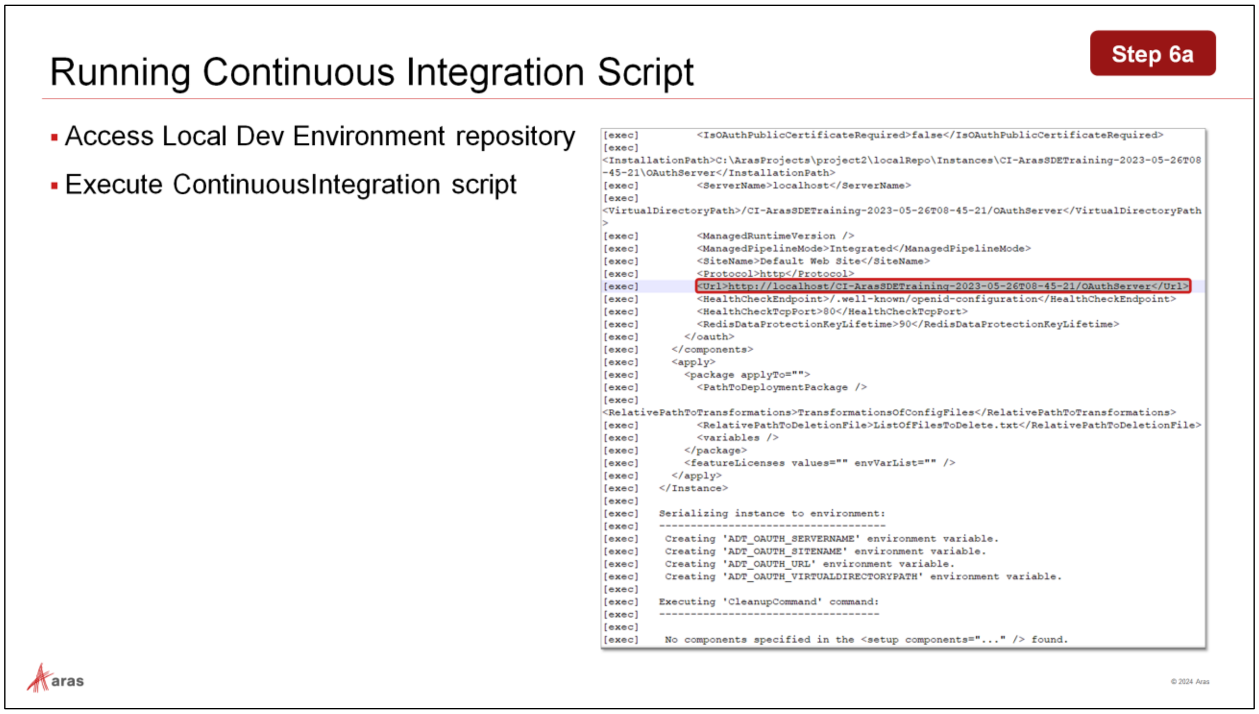
Continuous Integration Script

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).



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

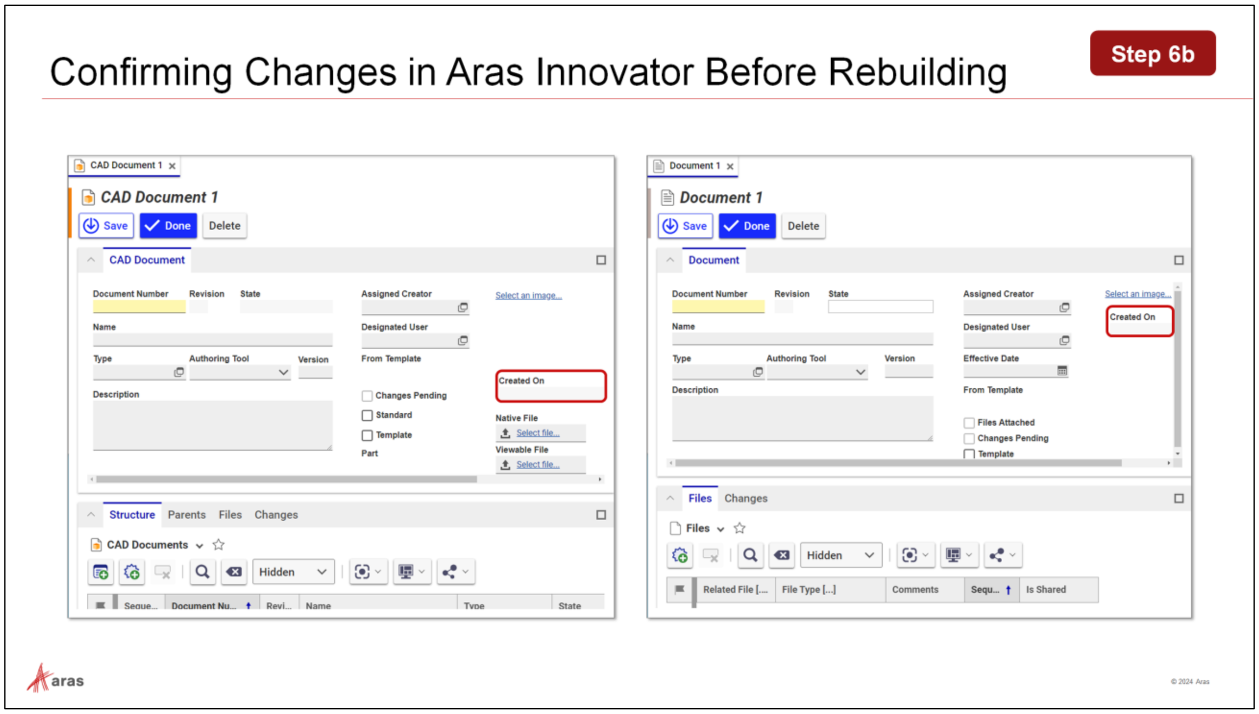
© 2024 Aras

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.

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

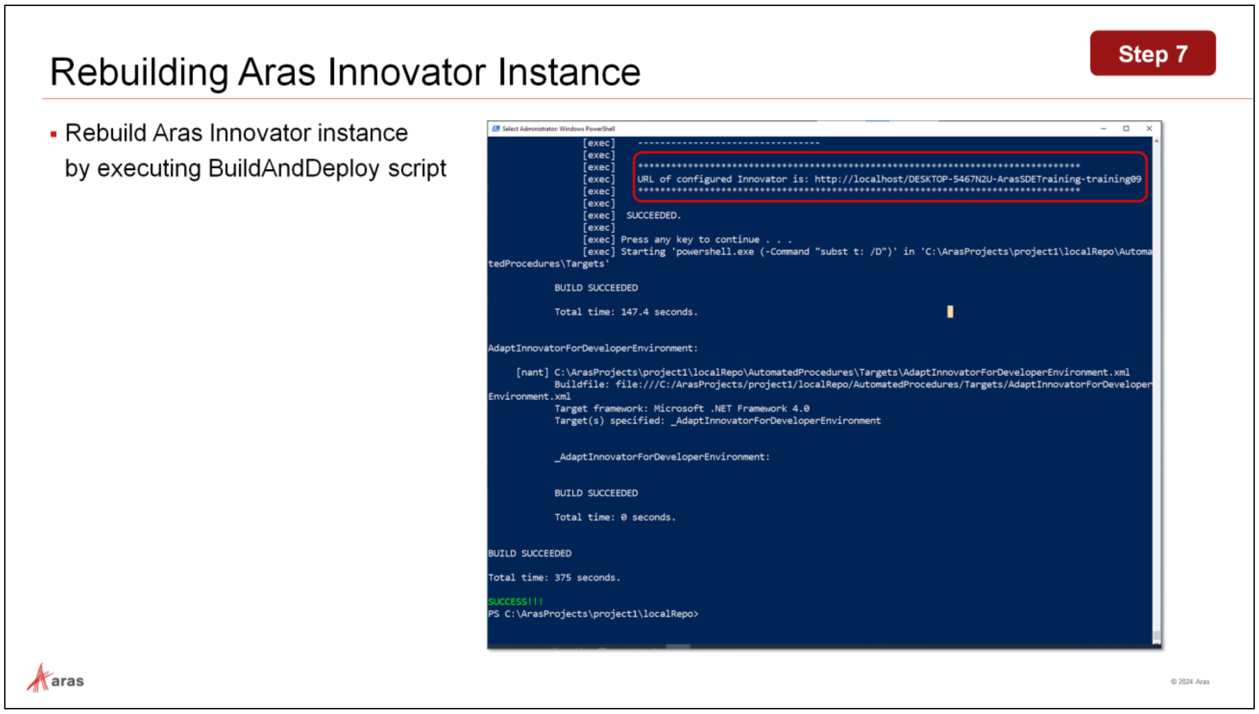
Step 6b

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

```
                [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>
```

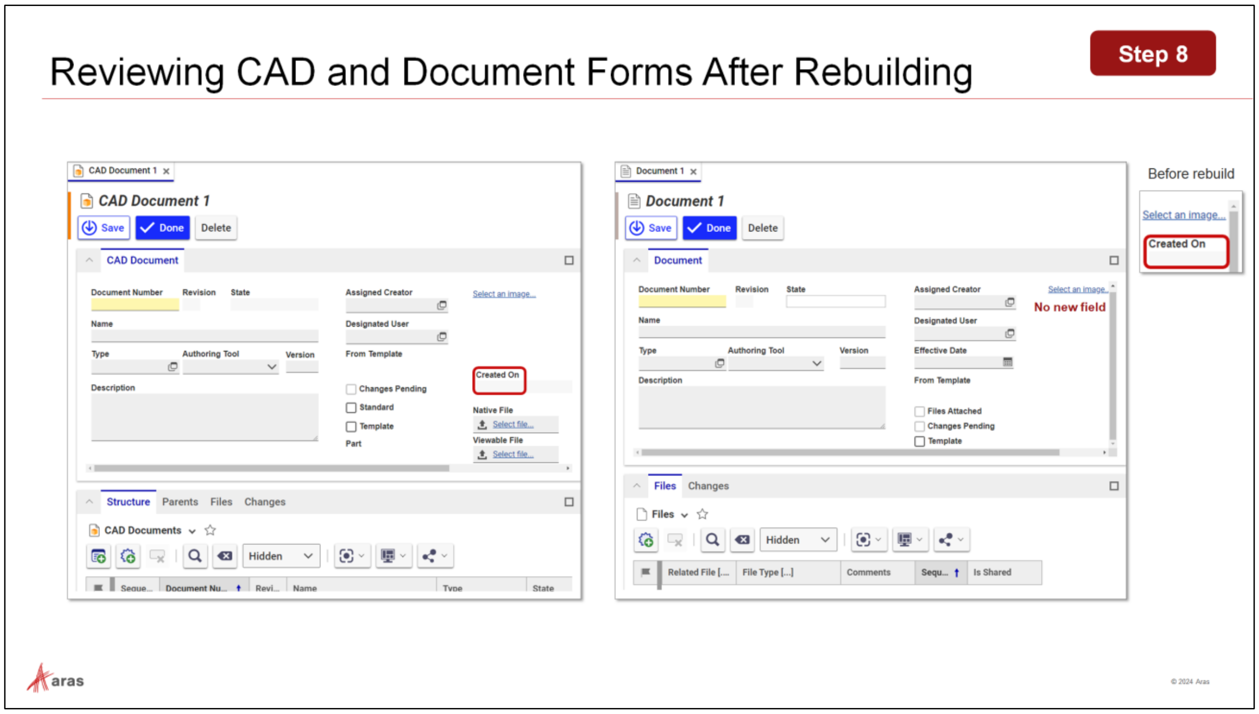
Rebuilding Aras Innovator Instance

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

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.

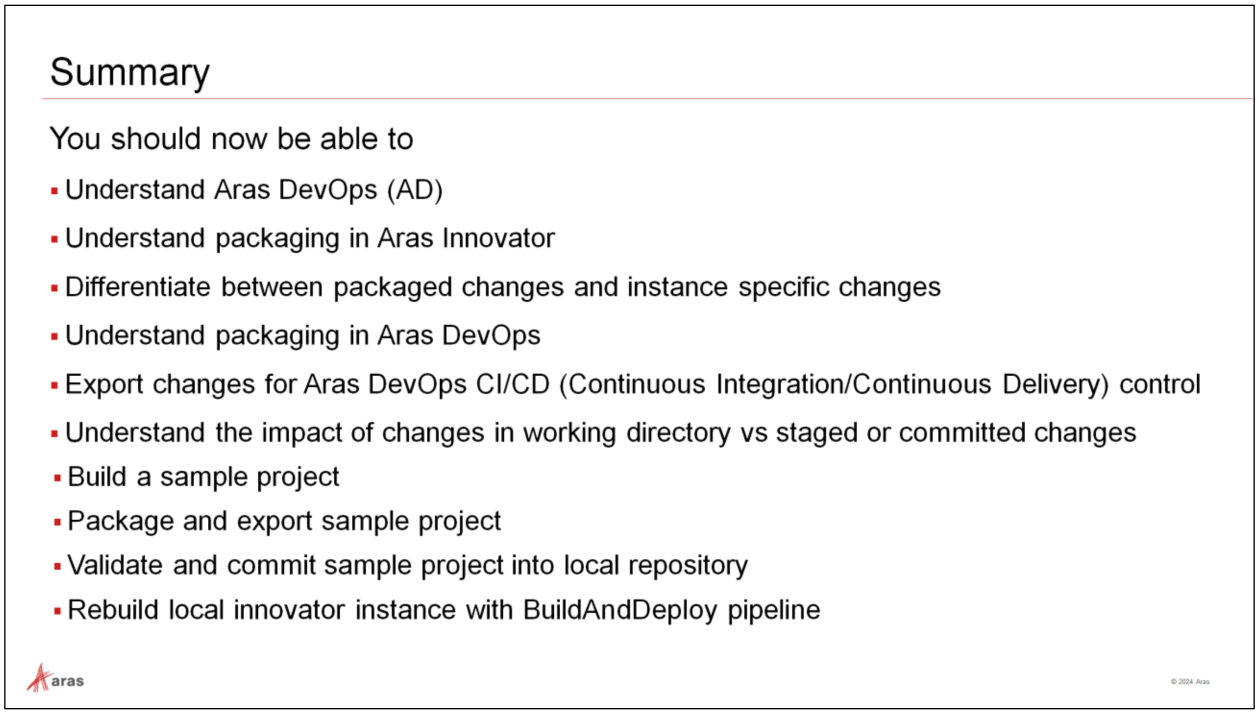
If you have kept your exports in separate folders, you can recover them.

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.



Summary

You should now be able to

- Understand Aras DevOps (AD)
- Understand packaging in Aras Innovator
- Differentiate between packaged changes and instance specific changes
- Understand packaging in Aras DevOps
- 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
- Build a sample project
- Package and export sample project
- Validate and commit sample project into local repository
- Rebuild local innovator instance with BuildAndDeploy pipeline

aras

Thank you for participating in this brief introduction to Aras DevOps CI/CD processes for Aras Innovator.

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