



aras

STUDENT
TRAINING GUIDE

Aras DevOps: Packaging and Continuous Integration

ACE23

REIMAGINE YOUR POSSIBILITIES

Copyright © 2023 by Aras Corporation. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, V1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

Distribution of substantively modified versions of this document is prohibited without the explicit permission of the copyright holder.

Distribution of the work or derivative of the work in any standard (paper) book form for a commercial purpose is prohibited unless prior permission is obtained from the copyright holder.

Aras Innovator, Aras, and the Aras Corp "A" logo are registered trademarks of Aras Corporation in the United States and other countries.

All other trademarks referenced herein are the property of their respective owners.

Microsoft, Office, SQL Server, IIS, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Notice of Liability

The information contained in this document is distributed on an "As Is" basis, without warranty of any kind, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or a warranty of non-infringement. Aras shall have no liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this document or by the software or hardware products described herein.

Revision MAY 2023

Packaging and Continuous Integration

Overview

In this session, you 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 and then export the project into the customer repository after validating it locally through 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

DevOps Defined

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



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

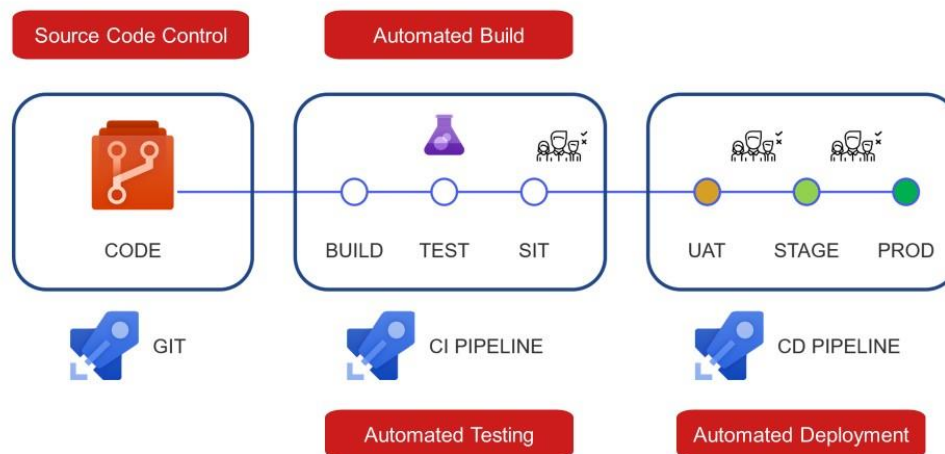


© 2023 Aras

DevOps Defined

- DevOps combines software development and IT operations to increase the efficiency, speed, and security of software development and delivery compared to traditional processes. A nimbler software development lifecycle results in a competitive advantage for businesses and their customers.
- DevOps is a practice that emphasizes the collaboration and communication of software developers and IT Operations. But it's more than that. It's a culture change. It's tearing down silos and breaking down walls to bring groups of people together to form a team with a common set of principles, methods, values, and tools.
- The process of application delivery and infrastructure management is automated. This is so that building, testing, and releasing software, as well as the deployment, configuration, and management of the infrastructure in which that software runs can happen rapidly, frequently, and more reliably. The team is accomplishing basic tasks planning, coding, building, testing, releasing, deploying, and configuring.
- But they do so in a continuous loop, a cycle of working class functionally. Together, many different members from technical teams, business teams, the coding team, production and deployment team, and the QA team all work through the entire process to understand and meet the needs of the customer with seamless IT services.
- CALMS: **C**ulture, **A**utomation, **L**ean, **M**easurement, **S**haring

Continuous Integration/Continuous Delivery (CI/CD) Explained



CI/CD Explained

DevOps and CI/CD tend to get used interchangeably:

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

CI/CD enables the delivery team to:

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

The illustration above shows the basic CI/CD flow:

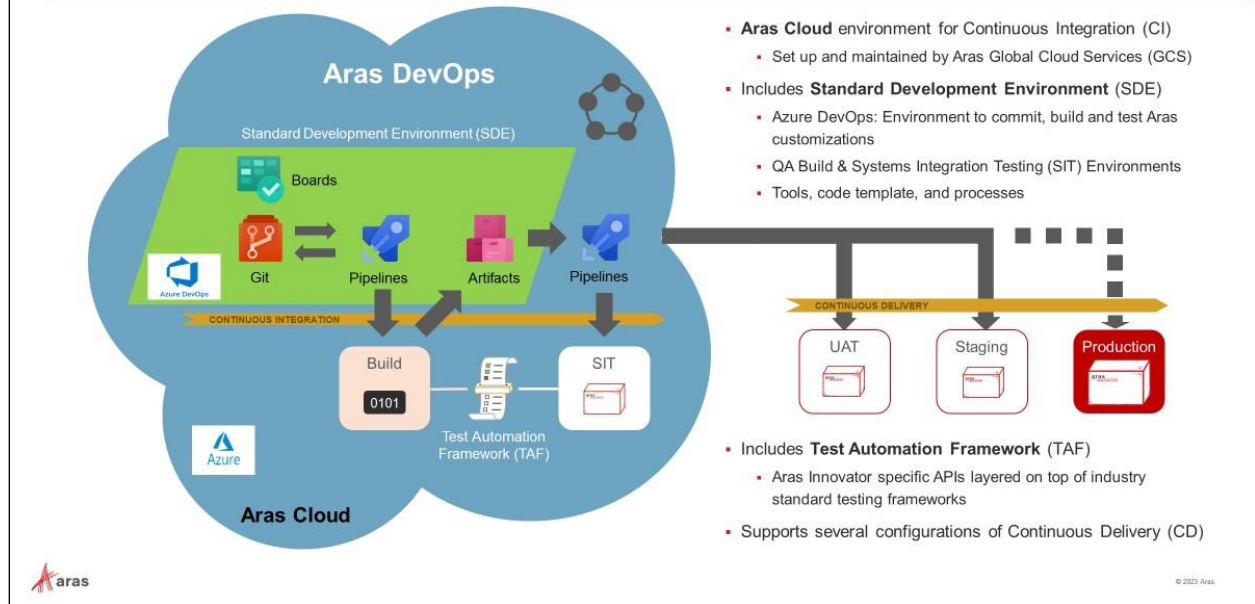
- Code = source code control. Source code control includes items such as configuration files and settings. It includes the code tree and the various libraries that constitute the solution.
- Commits are means to manage changes that take solution from one configuration to the next.
- The CI Pipeline supports the continuous integration where various contributors use pull requests (PRs) to submit their contributions to the integrated whole. The system automatically builds and runs available automated tests. Reviewers check the work before accepting it. The resulting artifacts are eventually deployed to SIT for automated testing.

- The CD Pipeline supports the need to deploy the validated (SIT tested) artifacts for user acceptance test and in some cases, it is used as a basis to verify the ability to migrate existing customer data into a staging area. Issues may arise during UAT. After remediation of the issues, the solution is eventually deployed into production.

Notes:

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

Aras DevOps High-Level View



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 (baselines, packages, builds) 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

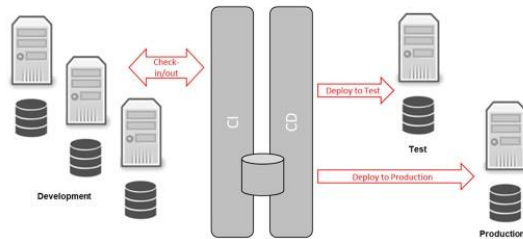
- Aras DevOps provides the full CI experience. Continuous Delivery is reviewed on a project-by-project basis to meet customer needs
- Aras Enterprise offering provides the full CI/CD offering for managing deployments as well as continuous integration.
- Full Continuous Delivery pipeline available instantly to support your DevOps Journey
- Continue to benefit as Aras evolves and introduces new enhancements into the Aras DevOps offering
- No additional internal hardware/maintenance required for supporting the pipeline
- Uses the same tools and process leveraged by Aras Solution Delivery teams
- 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 their 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 details that help 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**.



© 2023 Aras

Packaging in Aras Innovator

The Aras Innovator architecture is designed for customization of standard Aras Solutions/Apps and for building your own custom Solutions.

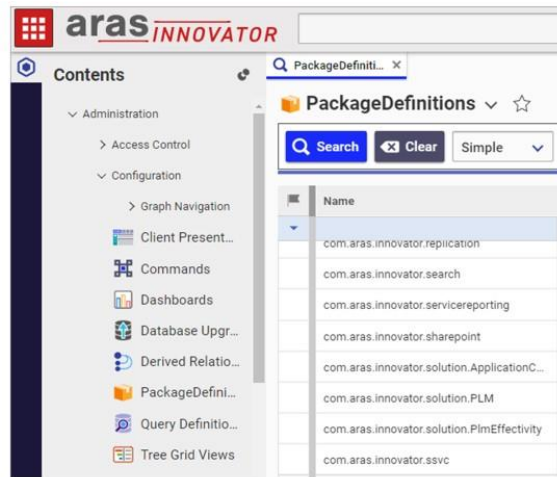
Solution Packaging is the mechanism that allows “Solution Developers” to register customizations in “Packages” so they can be extracted and transported to other Aras Innovator installations. For example, from a developer environment to the production system.

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 first step in moving a solution is to create a **Package Definition**.

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
- 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.



© 2023 Aras

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 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.
- **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!

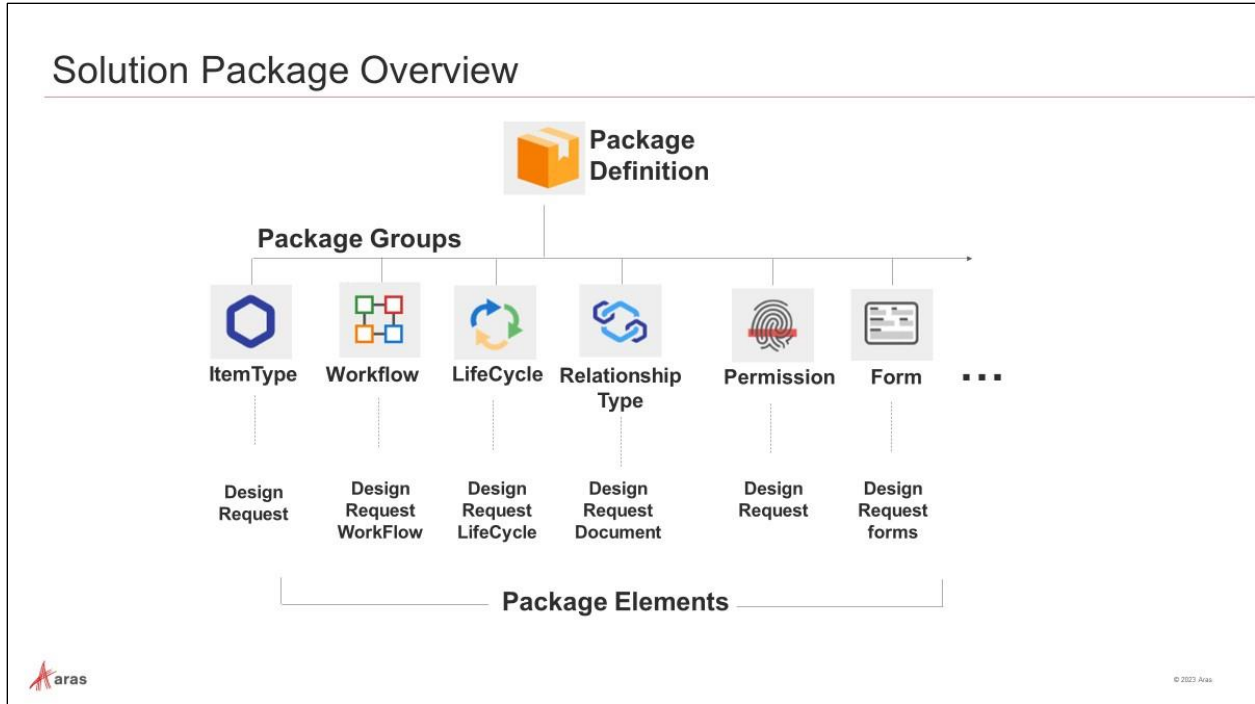
Try It ... Explore existing Package Definitions

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 name spaces defined in this convention are reserved for Aras internal packages and shall not be applied outside of Aras!



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 system they are defined in using the Packaging Utilities.

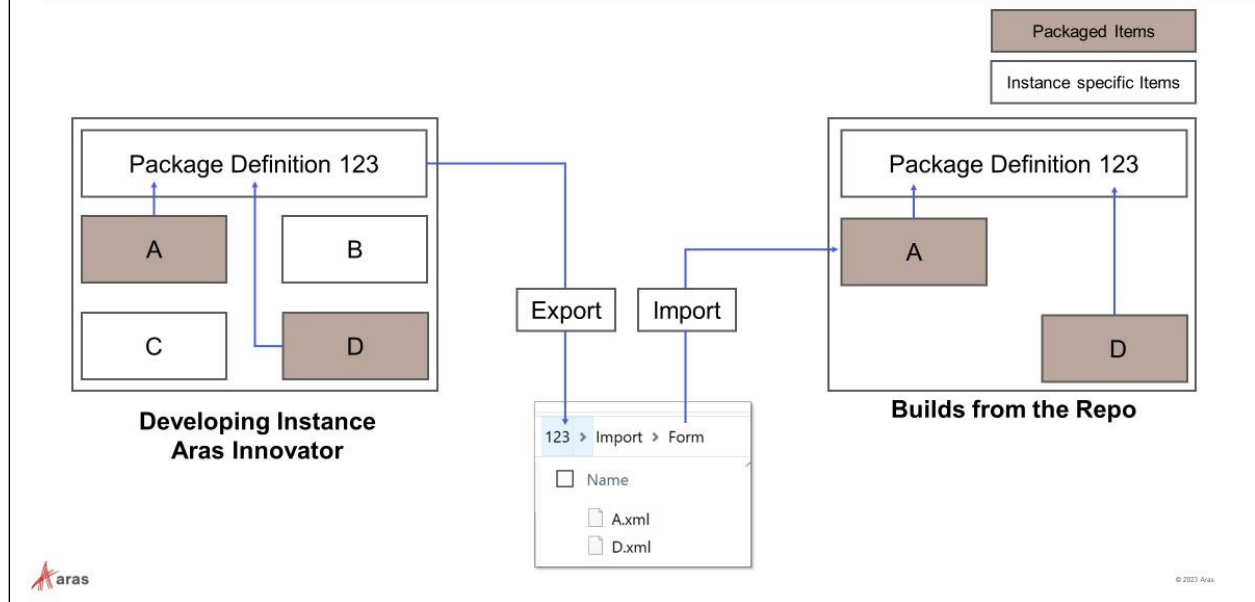
Package Groups

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 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



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 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.

When creating items, e.g., users, lists, test data, etc. in an Aras Innovator instance uniquely for using them in this instance, then packaging is not required. However, when exporting them for use in CI/CD, packaging is mandatory. This allows the build process to create an Innovator instance with information from source code repositories.

In the example above **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.

Please be aware that 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



© 2023 Aras

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.

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 example, we will change the CAD form and the Document form within the Aras solution package “com.aras.innovator.solution.PLM” package, which already exists in the baseline.

Do not move Elements between Packages

Once a package is released and got distributed, you should not move elements to other packages. If you do so, it will require scripts to be written to clean up references to the Package Elements you moved in Package Definitions in all environments the package is deployed in.

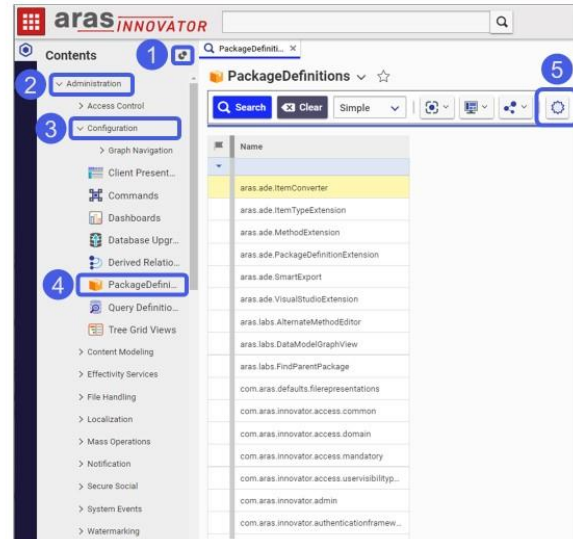
Definitions of dependencies between packages

- Often, the solutions you build reference elements from other solution packages. You can specify these package references as part of the Package Definition. When the exported package is imported into another system, the required dependencies are resolved before any new elements are accepted into the system.
- If a required package is missing on the destination system, an error is raised, and the import fails.

- Every solution in Aras Innovator is dependent on the Core Package. The Core Package is installed when you first install Aras Innovator. Because every solution relies on this package, the dependency is implied and does not have to be defined in a Package Definition. In the example above, the PLM Solution is dependent on Core, but that dependency is implied and does not have to be configured in the PLM Solution Package Definition.

Creating a Package Definition

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 in the PackageDefinitions group
5. Click on the create New PackageDefinition icon



Creating a Package Definition

You create a Package Definition to identify what elements will be exported from the database. You select the elements that make up your solution and add them to a definition. Eventually, the definition is used to export the selected Items from the database to the file system as AML documents.

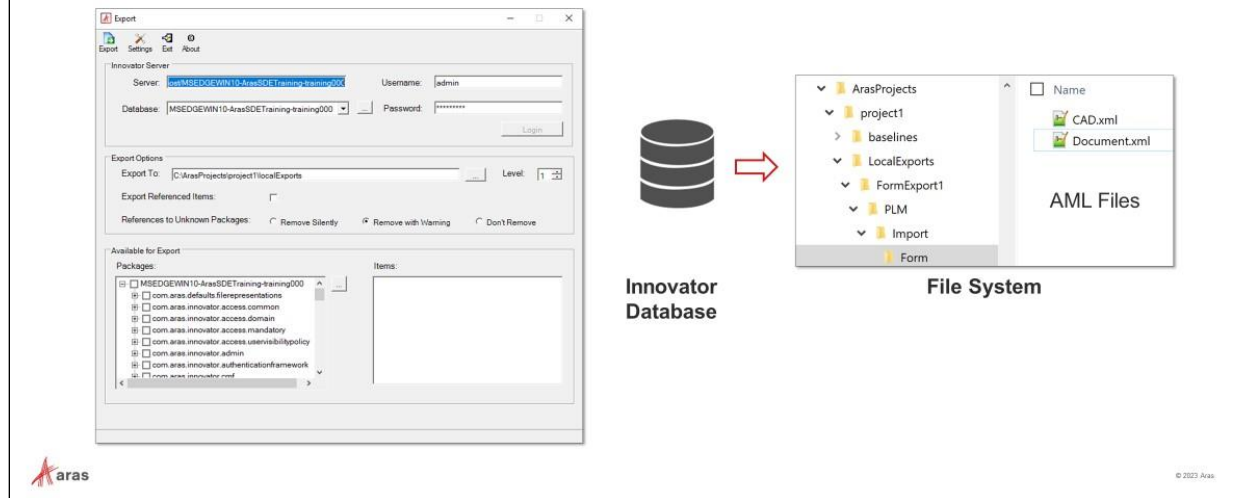
After creating a package, you can add subsequent elements by selecting each Item and adding them to the same Package Definition.

Try it ... Create a Package Definition

1. Navigate to **Administration > Configuration > PackageDefinitions** and click the **Create New PackageDefinition** button.
2. Enter *com.aras.training.sde* for the **PackageDefinition Name** and click **Done**.

Exporting a Package

- The Export utility may be downloaded from: <https://www.aras.com/en/support/downloads>.
- It allows you to export DB packages and their elements into file system.



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\IEUser\Desktop\Apps\PackageImportExportUtilities\Export` (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. For example: `http://localhost/MSEEDGEWIN10-ArasSDETraining-training`.
4. Click the ellipsis (...) button to the right of the **Database** field to populate the dropdown list.
5. Select the desired database, i.e., `MSEEDGEWIN10-ArasSDETraining-training`.
6. Enter a valid administrator username and password in the respective fields. In Aras classroom setting, use the **Username** of `admin` with the **Password** of `innovator`.
7. Click the **Login** button.

Use Case #1: CAD Form and Document Form Changes

	CAD Form	Document Form
Export before change	✓	✓
Change Form in the original Aras Innovator instance	✓	✓
Export after change	✓	✓
Copy export results to local repo	✓	✓
Stage/commit file	✓	✗
Run ContinuousIntegration	✓	✓
Run BuildAndDeploy	✓	✓
Changes in the new instance?	Yes ✓	No ✗



© 2023 Aras

Use Case #1: CAD Form and Document Form Changes

The objectives of the use case is to make changes to two items, already existing in the baseline, and to integrate the changes to the future builds: the CAD Form and the Document Form are the two items that we are modifying.

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

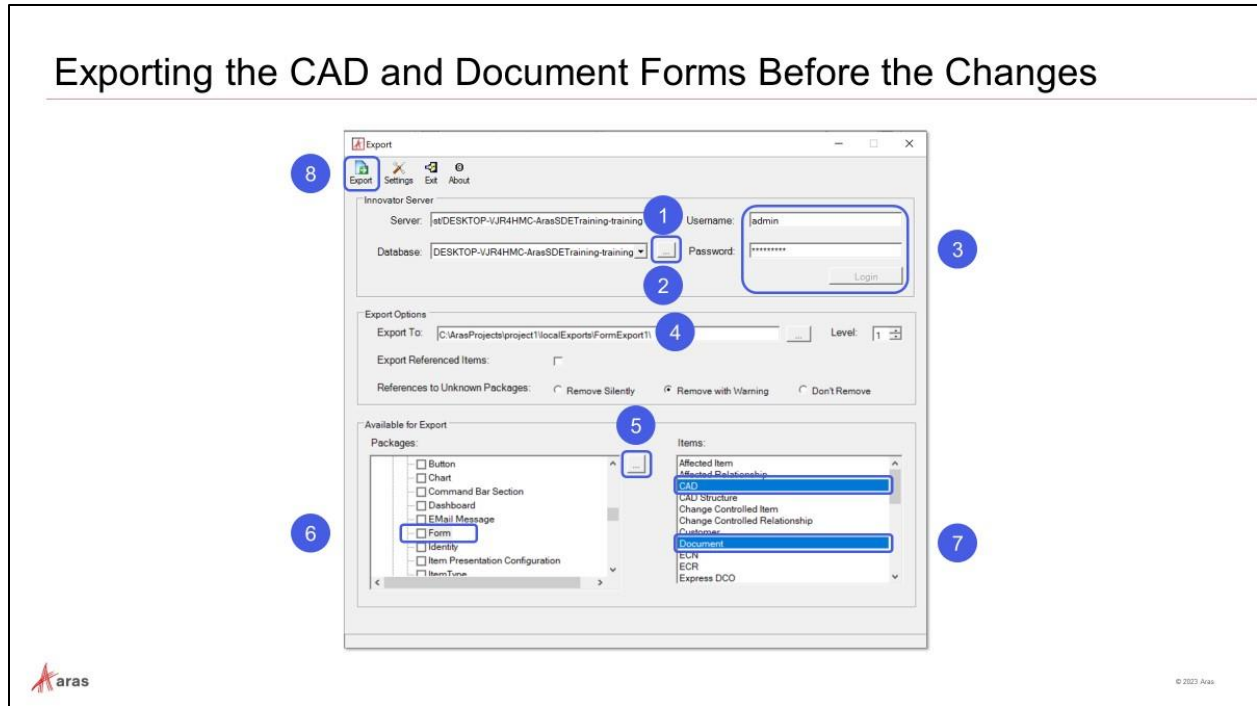
Since you have added the properties to existing items, there is no need to update the import manifest file in this exercise; we will not even have more package files than before, just two modified files.

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

The exercise steps contain checks and verifications which help to understand the details of the process as listed in the following table.

Step	CAD Form	Document Form	Result
Export before change	✓	✓	C:\ArasProjects\project\ localExports\FormExport1\
Change Form in the Aras Innovator instance	✓	✓	Forms in instance have one more field each
Export after change	✓	✓	C:\ArasProjects\project\ localExports\FormExport2\
View Differences	✓	✓	
Review the manifest file in the local repo and in the export results	✓	✓	The manifest file is not changed
Copy export results to local repo	✓	✓	Local repo contains the changes in the files
Stage or commit the xml file in the Form folder	✓	X	Only the modified file for the CAD form is in git
Run ContinuousIntegration	✓	✓	No error
Confirm change in git	✓	✓	git contains the changes as unstaged files
Confirm change in the Innovator instance	✓	✓	Forms in instance have still the new field each
Run BuildAndDeploy	✓	✓	No error
Confirm the changes in the new Innovator instance	✓	X	CAD: ok Document: like before the change

Exporting the CAD and Document Forms Before the Changes



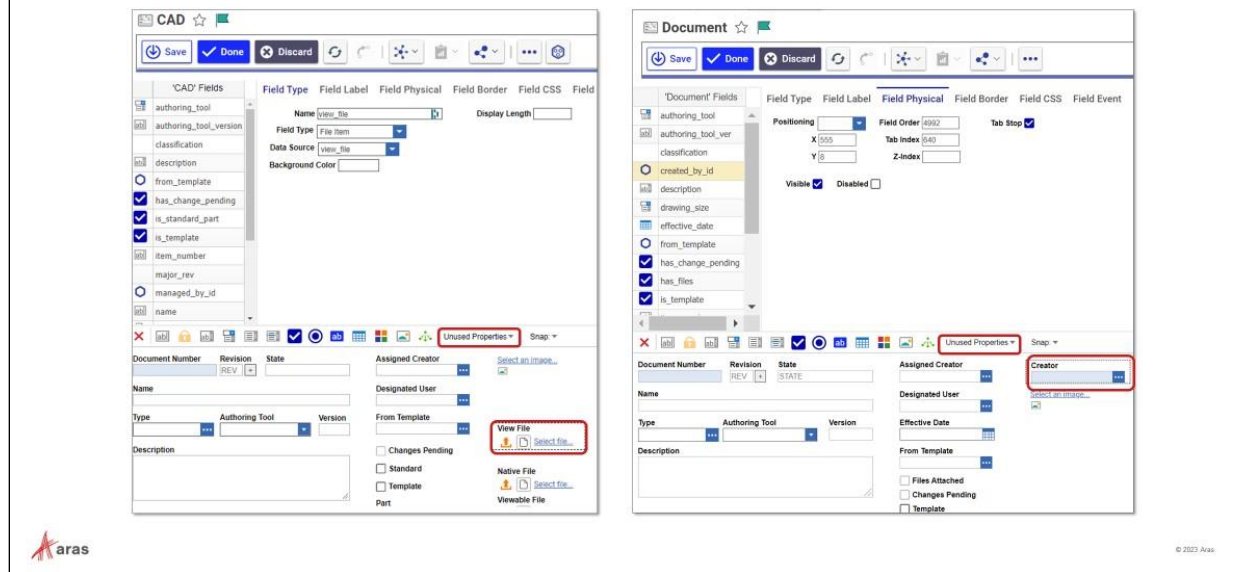
Exporting the CAD and Document Forms Before the Changes

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

Try It ... Export the CAD and Document Forms

1. Enter the server url.
2. Select the database (click on the three dotted = ellipsis button, then select from the drop-down).
3. Log in (fill in username admin and password, then click the **Login** button).
4. Set the destination of the export, you will find the exported files in that directory.
For example: *C:\ArasProjects\project1\localExports\FormExport1*.
5. Refresh packages (click on the three dotted = ellipsis button).
6. Locate the package group **Form** in the package definition **com.aras.innovator.solution.PLM** and click on the text.
7. Select the forms **CAD** and **Document**.
8. Click the **Export** button.
Note: A “**Create directory**” dialog will pop up asking you if you want to create a new directory. Click **Yes** to accept the offer to create the new directory.
9. Navigate to the newly created folder **FormExport1**, and review the **CAD.xml** and **Document.xml** files; also review the newly created manifest file **imports.mf**.
10. Review the export log in folder *..\PackageImportExportUtilities\Export\log*.

Changing the CAD and Document Forms



Changing the CAD and Document Forms

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

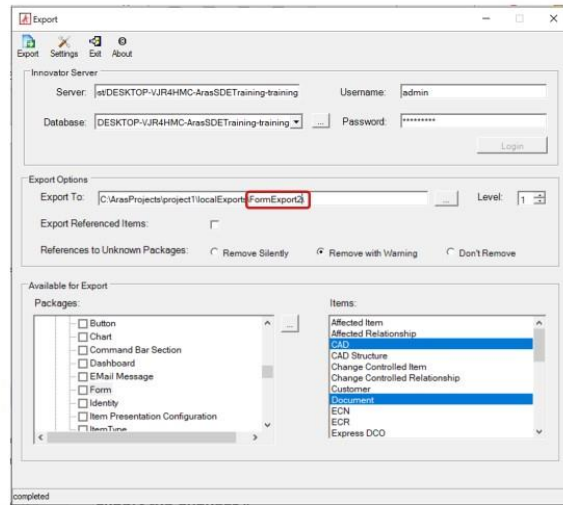
Try It ... Change the CAD Form

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 **view_file**.
5. Place the Field on the canvas, and click on **Done** to save the Form.

Try It ... Change the 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_by_id**.
5. Place the Field on the canvas, optionally change the **Field Label** to **Created By**, and click on **Done** to save the Form.

Exporting the CAD and Document Forms After the Changes



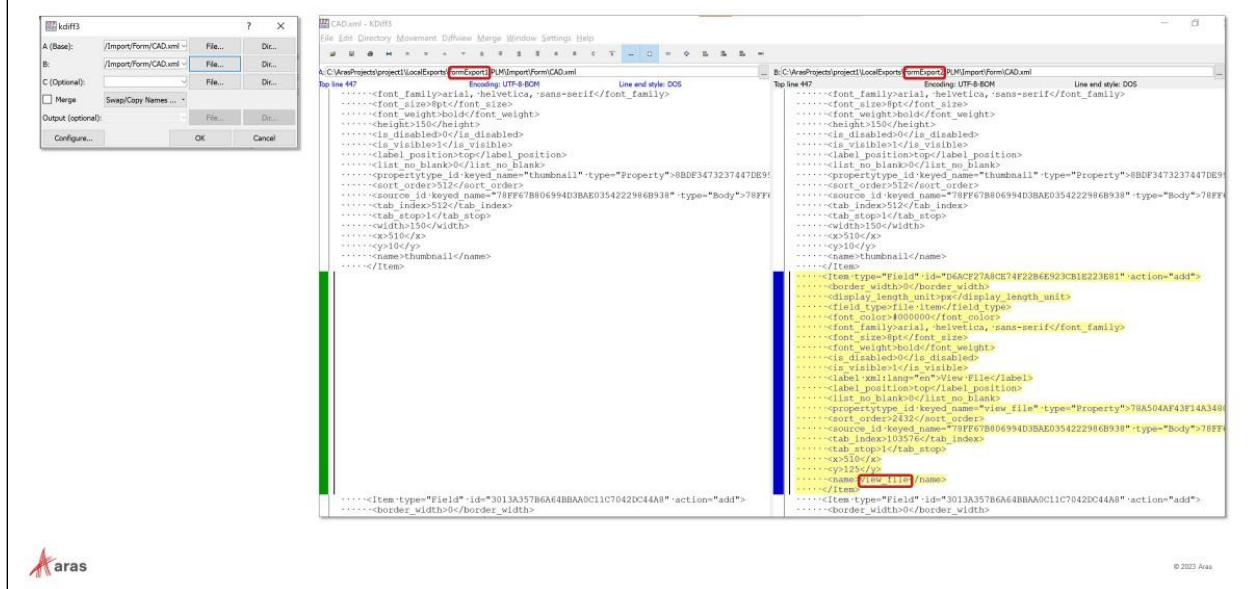
Exporting the CAD and Document Forms After the Changes

Export your changes to the new destination folder `C:\ArasProjects\project1\localExports\FormExport2`, marking the two Form items CAD and Document.

Try It ... Export the 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 log, and verify the contents of the folder **FormExport2**.

Locating the Differences (with KDiff3)



Locating the Differences (with KDiff3)

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

Try It ... Analyze Differences in the Result Files for the CAD Form

1. From the Taskbar or the Windows Start menu run KDiff3 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 **view_file** in the exported xml file after the change – that reflects exactly what we changed from the UI.

Try It ... Analyze Differences in the Result Files for the Document Form

1. From the Taskbar or the Windows Start menu run KDiff3 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: We find the definition of one additional field with the name property **created_by_id** in the exported xml file after the change – that reflects exactly what we changed from the UI.

Reviewing the Manifest File

- 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>
```



© 2023 Aras

Reviewing the Manifest File

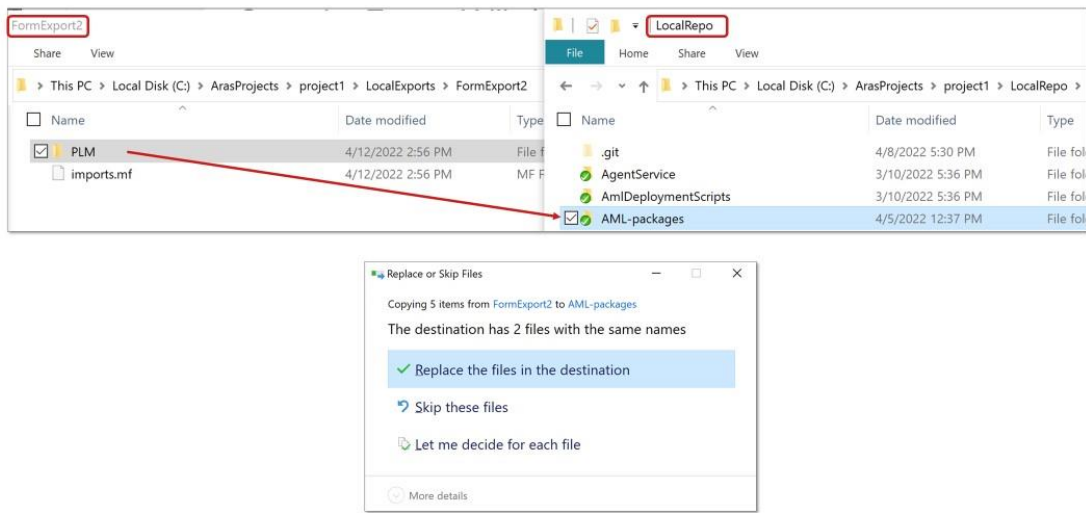
The manifest file is an important file that tells the CI/CD pipeline which packages need to be included during the import phase.

Try It ... Compare the Manifest Files in Export and AML-packages Folders:

1. Navigate to folder C:\ArasProjects\project1\localExports\FormExport1 and open imports.mf in Notepad++.
2. Navigate to folder C:\ArasProjects\project1\localRepo\AML-packages and open imports.mf in a new Notepad++ instance (used for future builds).
3. Both files will be identical in the PLM section.

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



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. In the **Replace or Skip Files** dialog select **Replace the files in the destination** to accept the replacement.

Staging the Modified File CAD.xml in Git Bash

```

1 ARAS-CORP+gvatteroth@TABLET-LCS5HJGK MINGW64 /c/ArasProjects/project1/LocalRepo/AML-packages/PLM
$ git status
On branch temp
Your branch is ahead of 'blessed/DevelopingSolutions' by 1 commit.
(use "git push" to publish your local commits)

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:   Import/Form/CAD.xml
        modified:   Import/Form/Document.xml

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

2 ARAS-CORP+gvatteroth@TABLET-LCS5HJGK MINGW64 /c/ArasProjects/project1/LocalRepo/AML-packages/PLM
$ git add Import/Form/CAD.xml

3 ARAS-CORP+gvatteroth@TABLET-LCS5HJGK MINGW64 /c/ArasProjects/project1/LocalRepo/AML-packages/PLM
$ git status
On branch temp
Your branch is ahead of 'blessed/DevelopingSolutions' by 1 commit.
(use "git push" to publish your local commits)

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

ARAS-CORP+gvatteroth@TABLET-LCS5HJGK MINGW64 /c/ArasProjects/project1/LocalRepo/AML-packages/PLM
$
  
```

Staging the Modified File CAD.xml in Git Bash

To add the modified file CAD.xml we stage it by using `git add ../CAD.xml`.

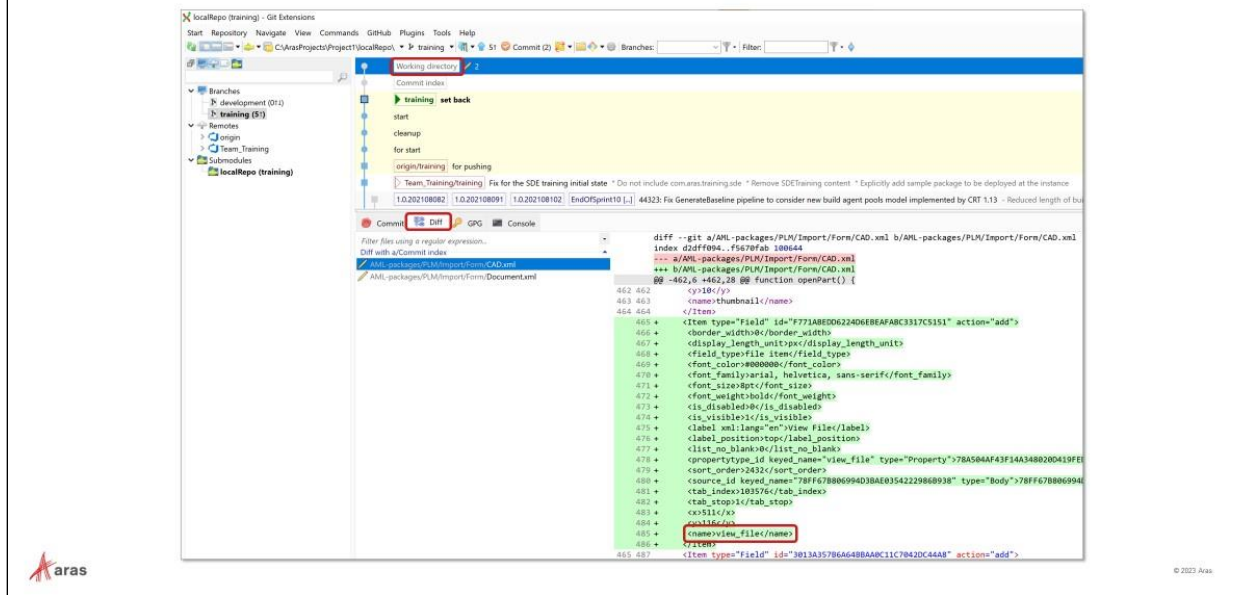
We do not stage the modified file Document.xml to see the different behavior.

Try It ... Stage the Modified File CAD.xml

1. Navigate to C:\ArasProjects\project1\localRepo\AML-packages\PLM, right-click to open the context menu and select **Git Bash Here**.
2. Verify the current status of the git repository by running `git status`.
Result: you will see the two files CAD.xml and Document.xml as not staged files (there are other files that are also not staged but they are irrelevant to use case).
3. Stage the file CAD.xml: `git add 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 modified file.
5. Commit the CAD.xml file using the command `git commit -m "Changed the CAD Form"`.

Note: The instructor will also demonstrate the steps above using the Git Extensions GUI.

Confirming Your Changes in Git Extensions



Confirming Your Changes in Git Extensions

The Git Extensions GUI allows one to quickly review modifications in files.

Try It ... Confirm Your Changes in Git

1. Navigate to C:\ArasProjects\project1\localRepo, right-click to open the context menu and select **GitExt Open repository**.
2. Click **OK** in the **Settings - Checklist** window.
3. Navigate to the tip of the **training** branch (below **Commit index**).
4. Click on the **Diff** tab and select the file CAD.xml.
5. Notice that it displays an additional Item block with the name property **view_file**.
6. Navigate to the **Working directory** (non-staged and non-committed work).
7. Click on the **Diff** tab and select the file Document.xml.
8. Notice that it displays an additional Item block with the name property **created_by_id**.

Reviewing the Continuous Integration Pipeline

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



© 2023 Aras

Reviewing the Continuous Integration Pipeline

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 locally 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.

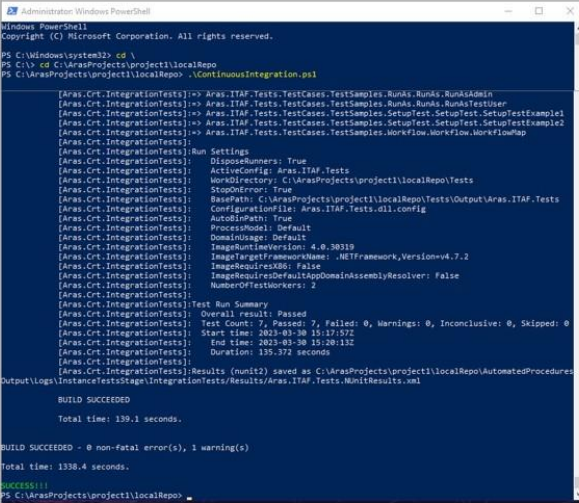
CI Steps

The CI script runs all the unit and integration tests that have been created for a project by installing and building a new temporary instance of Aras Innovator, applying the project code and configuration, and making sure all tests are successful.

A report indicates Success or Failure with an execution log that helps in resolving issues that may have occurred. In the final steps the script deletes the running instance (and database).

Running Continuous Integration

- Access local Dev Repository
- Execute ContinuousIntegration.ps1 script



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> cd \
PS C:\> cd C:\ArasProjects\project1\localRepo
PS C:\ArasProjects\project1\localRepo> .\ContinuousIntegration.ps1

[Aras.Crt.IntegrationTests]:> Aras.ITAF.Tests.TestCases.TestSamples.RunAs.RunAsAdmin
[Aras.Crt.IntegrationTests]:> Aras.ITAF.Tests.TestCases.TestSamples.RunAs.RunAsTestUser
[Aras.Crt.IntegrationTests]:> Aras.ITAF.Tests.TestCases.TestSamples.SetupTest.SetupTestExample1
[Aras.Crt.IntegrationTests]:> Aras.ITAF.Tests.TestCases.TestSamples.SetupTest.SetupTestExample2
[Aras.Crt.IntegrationTests]:> Aras.ITAF.Tests.TestCases.TestSamples.WorkFlow.WorkFlowMap
[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\project1\localRepo\Tests
[Aras.Crt.IntegrationTests]:   StopOnError: True
[Aras.Crt.IntegrationTests]:   BasePath: C:\ArasProjects\project1\localRepo\Tests\Output\Aras.ITAF.Tests
[Aras.Crt.IntegrationTests]:   ConfigurationFile: Aras.ITAF.Tests.dll.config
[Aras.Crt.IntegrationTests]:   AutoInPath: True
[Aras.Crt.IntegrationTests]:   ProcessMode: 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: 7, Passed: 7, Failed: 0, Warnings: 0, Inconclusive: 0, Skipped: 0
[Aras.Crt.IntegrationTests]: Start time: 2023-03-30 15:17:57Z
[Aras.Crt.IntegrationTests]: End time: 2023-03-30 15:20:13Z
[Aras.Crt.IntegrationTests]: Duration: 135.372 seconds
[Aras.Crt.IntegrationTests]:
[Aras.Crt.IntegrationTests]: Results (html) saved as C:\ArasProjects\project1\localRepo\AutomatedProcedures\
Output\Logs\InstanceTestsStage\IntegrationTests\Results\Aras.ITAF.Tests.MUnitResults.xml

BUILD SUCCEEDED
Total time: 139.1 seconds.

BUILD SUCCEEDED - 0 non-fatal error(s), 1 warning(s)
Total time: 139.4 seconds.
SUCCESS!!!
PS C:\ArasProjects\project1\localRepo>
```



© 2023 Aras

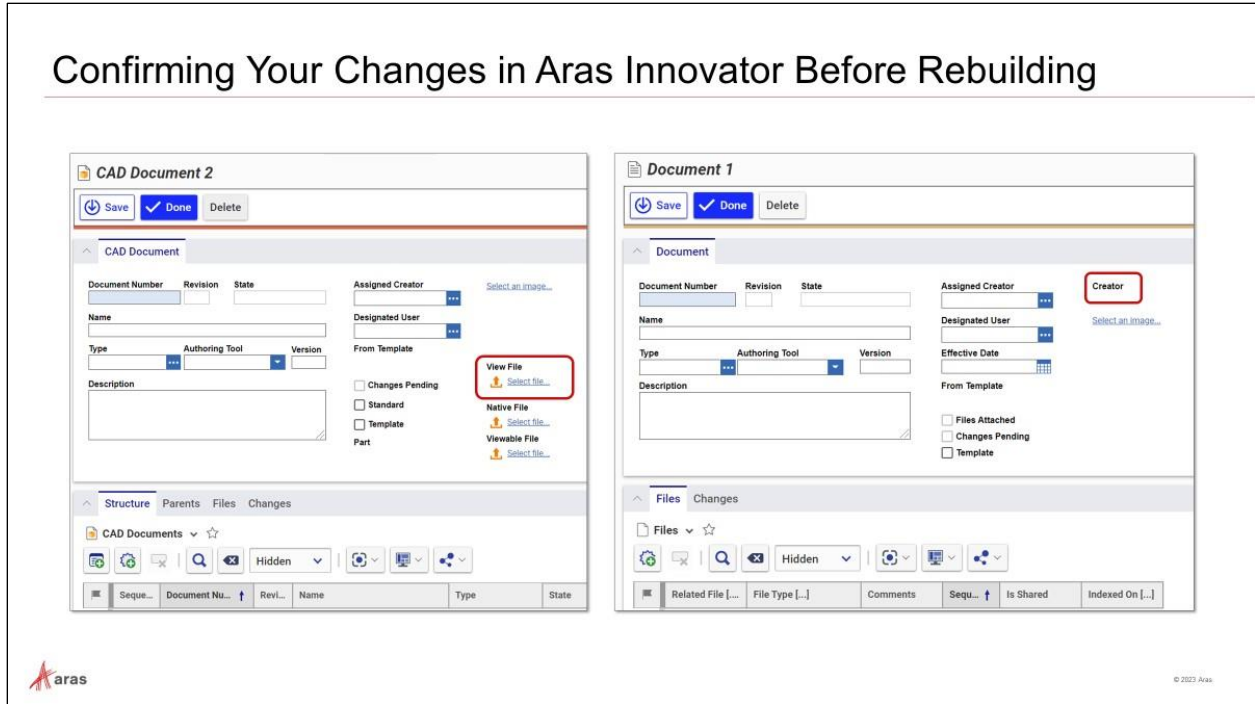
Running Continuous Integration

Run the ContinuousIntegration.ps1 file as administrator to ensure that your repository is still in a valid state. You should receive a green success message. In the example above the run took around 20 minutes.

Try It ... Run Continuous Integration

1. Right-click on the **Start** menu to open a new Windows PowerShell (Admin) window.
2. Access the local repository at C:\ArasProjects\project1\localRepo.
3. Enter `.\ContinuousIntegration.ps1` to execute the CI script.

Confirming Your Changes in Aras Innovator Before Rebuilding



Confirming Your Changes in Aras Innovator Before Rebuilding

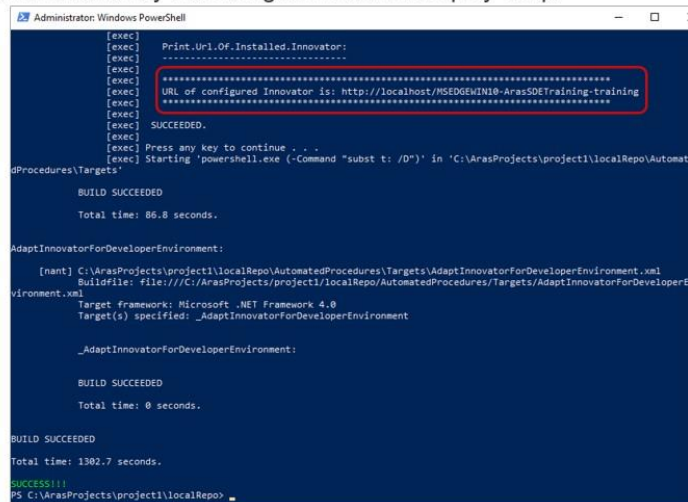
Verify that the current State instance is using the modified forms for CAD and Document items. In the next steps we will delete the current installation and create a new one.

Try It ... Confirm Your Changes in Aras Innovator Before Rebuilding

1. In the TOC go to **Documents** to create the new documents.
2. Go to **Documents > CAD Documents** to create a CAD Document: notice the new field named **View File**.
3. Go to **Documents > Documents** to create a Document: notice the new field named **Created By** (or `created_by_id` if you did not do the optional renaming).

Rebuilding Aras Innovator

- Rebuild Aras Innovator instance by executing the BuildAndDeploy script



```
Administrator: Windows PowerShell
[exec]
[exec] Print.Url_of_Installed_Innovator;
[exec] -----
[exec] *****
[exec] URL of configured Innovator is: http://localhost/MSEDEMIN10-ArasSDETraining-training
[exec] *****
[exec]
[exec] SUCCEEDED.
[exec]
[exec] Press any key to continue . . .
[exec] Starting 'powershell.exe (-command "subst t: /D")' in 'C:\ArasProjects\project1\localRepo\Automate
dProcedures\Targets'
BUILD SUCCEEDED
Total time: 86.8 seconds.

AdaptInnovatorForDeveloperEnvironment:
[nant] C:\ArasProjects\project1\localRepo\AutomatedProcedures\Targets\AdaptInnovatorForDeveloperEnvironment.xml
Buildfile: file:///C:/ArasProjects/project1/localRepo/AutomatedProcedures/Targets/AdaptInnovatorForDeveloperEn
vironment.xml
Target framework: Microsoft .NET Framework 4.0
Target(s) specified: _AdaptInnovatorForDeveloperEnvironment
_AdaptInnovatorForDeveloperEnvironment:
BUILD SUCCEEDED
Total time: 0 seconds.

BUILD SUCCEEDED
Total time: 1302.7 seconds.
SUCCESS!!!
PS C:\ArasProjects\project1\localRepo>
```



© 2023 Aras

Rebuilding Aras Innovator

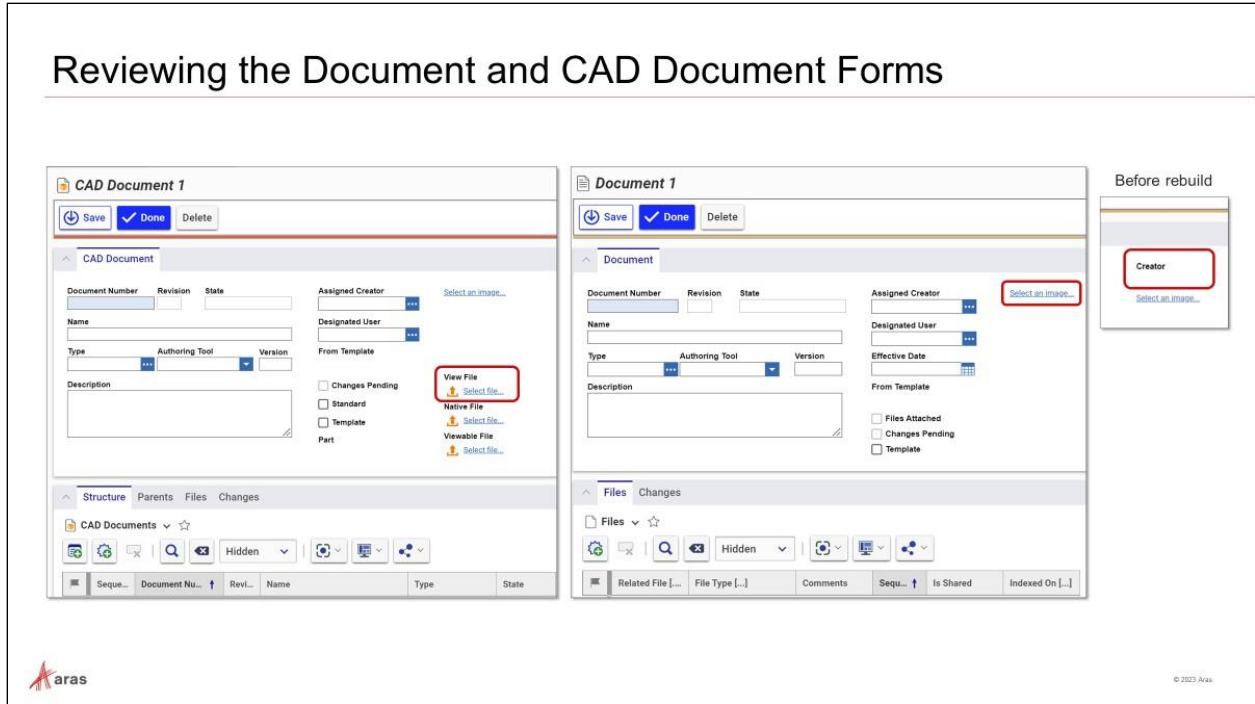
Rebuild Aras Innovator using the script BuildAndDeploy.ps1. You should receive a green success message. In the example above the script execution took around 20 minutes.

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

Try It ... Rebuild Aras Innovator

- Right-click on the **Start** menu to open a new Windows PowerShell (Admin) window.
- Access the local repository at C:\ArasProjects\project1\localRepo.
- Enter **.\BuildAndDeploy.ps1** to execute the Build script.

Reviewing the Document and CAD Document Forms



Reviewing the Document and CAD Document Forms

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 the creator.

The example for the document form illustrates what happens if you have not staged or 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 the CAD and Document Forms

1. Login to Aras Innovator as user admin.
2. Select **Administration > Forms** and search for the CAD Document Form and Document Form.
3. Open the forms and notice any new changes.

Use Case #2: New ItemType Project

▪ Project Goal

- Create a simple Design Request to collect user data
- Export changes to project package
- Rebuild Innovator instance with new configuration

▪ Development Requirements

- Create ItemType
- Create Properties
- Create Sequence
- Configure Form
- Package Name: `com.aras.training.sde`
- Package content prefix: `trn_`

The screenshot shows a form titled "Design Request" with the following fields:

Created By	Title	
Created On	Completion Date	Patent?
	10/18/2017	<input type="checkbox"/>



© 2017 Aras

Use Case #2: New ItemType Project

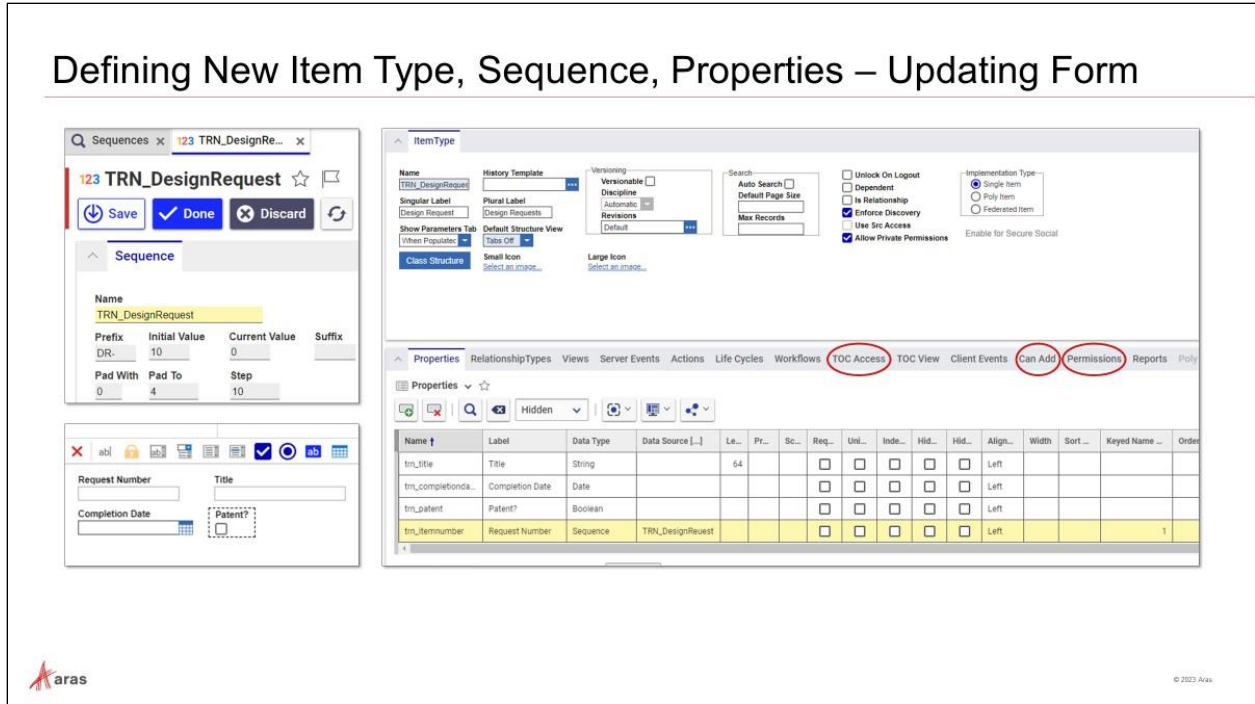
To demonstrate how the development process works, a small project has been designed, and this will allow users to collect information in a Design Request item.

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)

Requirement: The Design Request item should display the three properties on the form.

Defining New Item Type, Sequence, Properties – Updating Form



Defining New Item Type, Sequence, Properties – Updating Form

The following are the objectives:

1. Create a new **Item Type** named *trn_DesignRequest* with the appropriate labels.
2. Create a new **Sequence** named *trn_DesignRequestSequence* that will be used in the steps below to define the request number (DR-0010, DR-0020, etc.)
3. Create the following properties to collect data about the request:

Property	Label	Data Type	Length	Keyed Name Order
trn_title	Title	String	64	
trn_completionDate	Completion Date	Date		
trn_patent	Patent?	Boolean		
trn_itemNumber	Request Number	Sequence		1

4. Add the properties to the Design Request Form.
5. Assign TOC Access and Can Add? to the World identity.
6. Assign Default Access as the default permission.

Try it ... Create the Design Request ItemType

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

Name	trn_DesignRequest
Singular Label	Design Request
Plural Label	Design Requests
History Template	Default
Default Structure View	Tabs Off

Small/Large Icons	Choose from provided image list
--------------------------	---------------------------------

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

Try it ... Provide Access Permissions to the Design Request ItemType

- Provide the following values by clicking on each **Relationship** tab and adding a new row (the **Add** button):

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

- Save the record.

Try it ... Create a Sequence

- Navigate to **Administration > Sequences** and create a new **Sequence**.
- Name the sequence *trn_DesignRequestSequence*.
- Provide the following sequence values:

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

- Click the **Done** button to save the *trn_DesignRequestSequence* sequence.
- Return to the *trn_DesignRequest* **ItemType** tab and ensure you are in **Edit** mode.
- On the **Properties** tab, click the **New Property** button.
- Add a new property named *trn_itemNumber*, with a Label of *Request Number*, and with Data Type **Sequence** that uses the *trn_DesignRequestSequence* as the **Data Source**.

Name	Label	Data Type	Data Source [...]	KNO
<i>trn_itemNumber</i>	Request Number	Sequence	<i>trn_DesignRequestSequence</i>	1

- Add a Keyed Name Order (KNO) value of *1* for the **trn_itemNumber** property.
- Click the **Save** button to save the new property.
- Remain in **Edit** mode for the next exercise.

Try it ... Create Custom Properties for the Design Request

- Open the **Design Request** ItemType for editing.
- Select the **Relationship** tab labeled **Properties** and click the **New Property** button to add a new property.
- 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	KNO
trn_title	Title	String	64	
trn_completionDate	Completion Date	Date		
trn_patent	Patent?	Boolean		

4. Click **Done** on the Design Request ItemType.

Try it ... Regenerate the Design Request Form

1. Navigate to **Administration > ItemTypes** and open the **Design Request** ItemType for viewing.
2. On the **Views** relationship tab, right-click the Design Request (Default) Form and select **Actions >RebuildViewAction** from the context menu.
This will add all custom Design Request properties to the Form.
3. Right-click the Design Request (Default) Form again and select **Open** to open the Form Editor.
Notice that all custom properties have been added to the Form as Fields, i.e., the Form has been regenerated. You can reposition each Field by dragging them and set Field characteristics using the Field tabs.

Try it ... Add Design Request ItemType Button to TOC

We need to add the new Design Request ItemType to the **TOC Editor** to enable users to access the new Design Request Item from the TOC, and in the Main Search Grid.

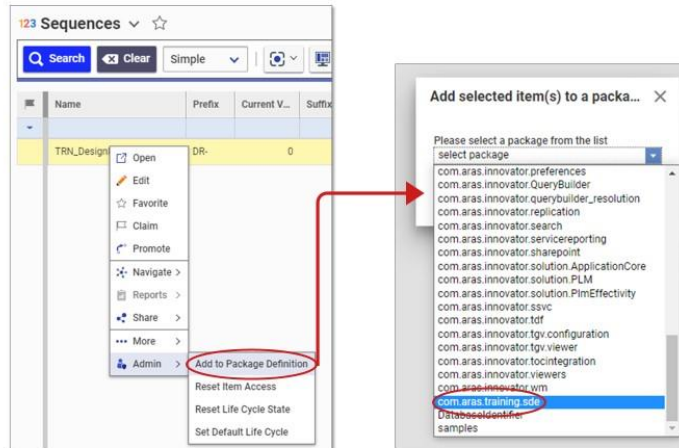
The Add ItemType button adds a new ItemType button to the TOC. If a Category is selected when this button is clicked, the new ItemType button is created as a child of the selected Category. If no Category is selected, the new ItemType button is created at the bottom of the TOC.

1. Navigate to **Administration > Configuration > TOC Editor** and select the **Design** category.
2. In the toolbar click the **Add ItemType** button.
3. Search for and select the *Design Request* ItemType and click **OK**.
4. Accept the default **Label** value and for the **Access** field, enter *World* or search for and select *World*.
5. Click **Save**. , and verify that the *Design Request* Item shows up under the **Design** category in the TOC.

Adding Items to the Export Package

Add project elements to package `com.aras.training.sde`

- Form
- ItemType
- Sequence



Adding Items to the Export Package

Add project elements to package `com.aras.training.sde`.

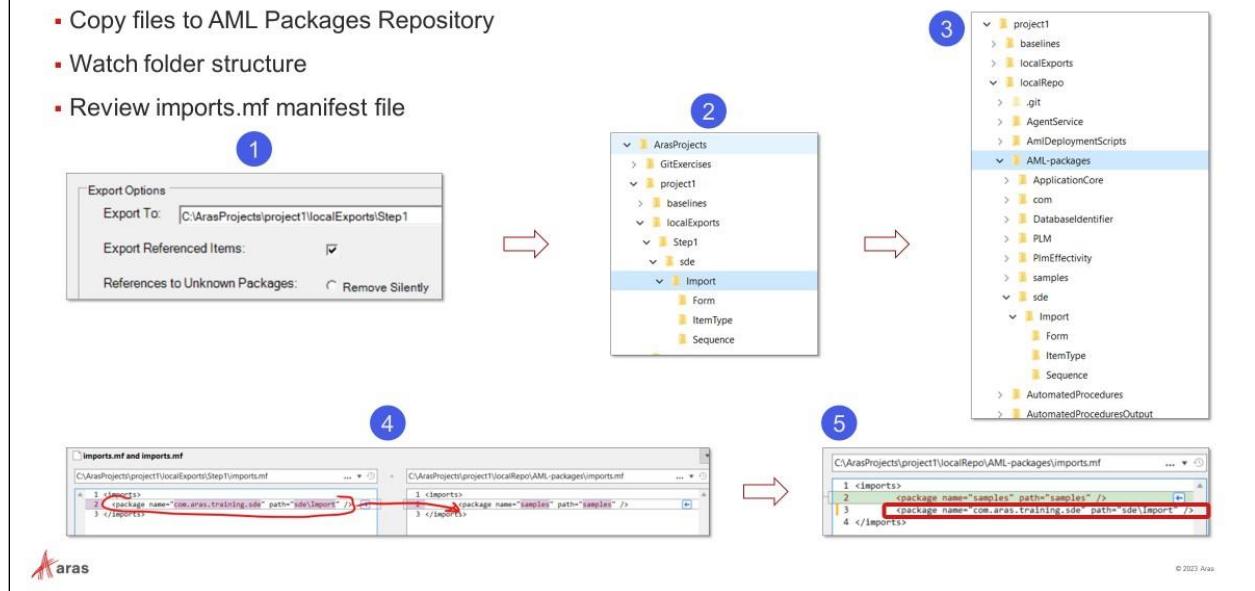
- Form
- ItemType
- Sequence

Try It Add Items to Export Package

1. Navigate to **Administration > ItemTypes** and search for `trn_DesignRequest`.
2. In the Main Search grid, right-click on `trn_DesignRequest`, and select **Admin > Add to Package Definition** from the context menu.
3. Search for and select `com.aras.training.sde`.
4. Repeat the steps for the other project elements.
5. Open the `com.aras.training.sde` package definition and verify its contents.

Placing Content Into a Repository

- Copy files to AML Packages Repository
- Watch folder structure
- Review imports.mf manifest file



Placing Content Into a Repository

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

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

1. Copy and paste the folder C:\ArasProjects\project1\localExports\Step1\sde into C:\ArasProjects\Project1\localRepo\AML_packages
2. Accept prompts for replacement if encountered.
3. Ensure that the imports.mf file includes the following line:
`<package name="com.aras.training.sde" path="sde\import" />`

Updating the Imports Manifest File

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



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



© 2023 Aras

Updating the Imports Manifest File

The previous example was demonstrating the quite simple case of modification of items which already exist in the baseline.

For setting up a module of your own you would normally create your own package by first creating your own package definition with its dependencies and then export the package.

In that case the manifest file for the import must be adapted and needs to be staged and committed in Git.

If you forget to update the manifest file that has the effect of ignoring your changes in the re-import and therefore it will not be considered for the following builds. **ContinuousIntegration.ps1** cannot help here as there are no syntactic errors to show.

Try It ... Update the Imports Manifest File

1. Open the manifest file that was created by the Export utility in Notepad++. You will find inside the line `<package name="com.aras.training.sde" path="sde\Import" />`.
2. Open the manifest file in the folder `C:\ArasProjects\project1\localRepo\AML-packages`, in a different Notepad++ instance.
3. Copy the line from step 1 into the file from step 2.
4. Save and close the **imports.mf** file from the AML-packages folder.

Reviewing ItemType AML Definition

- In order to avoid issues, open ItemType\TRN_DesignRequest.xml
- Check that there is no ITPresentationConfiguration exported
- If you see this node in your XML comment it out it

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <aml>
3 <item type="ItemType" id="7106A9F8EBAB439F806E37CC37EFDCE" action="add">
4 <allow_private_permission>1</allow_private_permission>
5 <auto_search>0</auto_search>
6 <enforce_discovery>1</enforce_discovery>
7 <hide_where_used>0</hide_where_used>
8 <history_template keyed_name="Default" type="History Template">3BC16EF9E52B4F9792AB76BCB0492F29</history_template>
9 <implementation type="table">/implementation_type</implementation_type>
10 <instance_data>TRN_DESIGNREQUEST</instance_data>
11 <is_dependent>0</is_dependent>
12 <is_relationship>0</is_relationship>
13 <is_versionable>0</is_versionable>
14 <label xml:lang="en">Design Request</label>
15 <label_plural xml:lang="en">Design Requests</label_plural>
16 <large_icon>./images/Activity2.svg</large_icon>
17 <open_icon>./images/Activity2.svg</open_icon>
18 <revisions keyed_name="Default" type="Revision">7FE395DD8B9F4E1090756A34B733D75E</revisions>
19 <show_parameters_tab>1</show_parameters_tab>
20 <structure view>tabs off</structure view>
21 <unlock_on_logout>0</unlock_on_logout>
22 <use_src_access>0</use_src_access>
23 <name>trn_DesignRequest</name>
24 <relationships>
25 <!--<item type="ITPresentationConfiguration" id="9AD21FBCA33C4D20AD37A75DEDFE543F" action="add">
26 <client>js</client>
27 <related_id keyed_name="trn_DesignRequest_TOC_Configuration"
28 type="PresentationConfiguration">0582E507F1EB4EFAA70C76FAD063AAAF</related_id>
29 <sort_order>128</sort_order>
30 <source_id keyed_name="trn_DesignRequest" type="ItemType" name="trn_DesignRequest">7106A9F8EBAB439F806E37CC37EFDCE</source_id>
31 </item-->
32 </relationships>
33 </item>-->
34 <item type="Property" id="72F888E99A174F9E95696541ECF7C63" action="add">
```



© 2023 Aras

Reviewing ItemType AML Definition

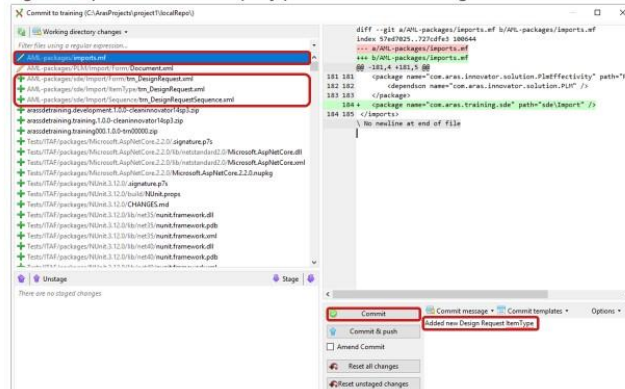
Check and comment out the <ITPresentationConfiguration> node from your ItemType definition. It is a known issue: if left here then we will fail later when we carry out the building of the new Innovator instance.

Try It ... Review the ItemType AML Definition

1. Go to C:\ArasProjects\Project1\localRepo\AML_packages\sde\ItemType.
2. Open the trn_DesignRequest.xml file in Notepad++.
3. Comment out the node <ITPresentationConfiguration> if encountered.

Staging and Committing Your Changes

- Review changes in the working repo
- Stage changes
- Commit your changes with appropriate and informative message
- (Execute ContinuousIntegration.ps1/BuildAndDeploy.ps1 → ensure it is green and that everything is in place)



Reminder:
If you run **BuildAndDeploy** in this state (before atage/commit), you will wipe out all your changes from Innovator instance.

Staging and Committing Your Changes

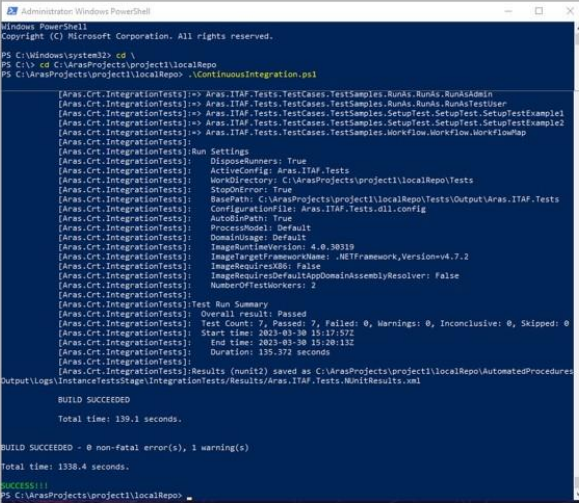
It is preferable to do atomic commits to better control your work; for this reason, it is advised to include code (such as validation methods) in other commits.

Try It ... Stage and Commit Your Changes

1. Navigate to C:\ArasProjects\project1\localRepo and open Git Extensions.
2. Click on **Commit** in the middle of the main toolbar.
3. In the **Working directory changes** pane, you will notice the **four** files we are interested in staging and committing.
4. Select the four different files, i.e., imports.mf, trn_DesignRequest.xml (2) and trn_DesignRequestSequence, and transfer them to the **Stage** area.
5. Enter a commit message, for instance *“Added new Design Request ItemType”*, and click on **Commit**.
6. Verify now the new status of the Git repository by running *git status* from a Git Bash window.

Validating Build

- Access local Dev Repository
- Execute ContinuousIntegration.ps1 script



```
Administration: Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> cd \
PS C:\> cd C:\ArasProjects\project1\localRepo
PS C:\ArasProjects\project1\localRepo> .\ContinuousIntegration.ps1

[Aras.Crt.IntegrationTests]:> Aras.ITAF.Tests.TestCases.TestSamples.RunAs.RunAsAdmin
[Aras.Crt.IntegrationTests]:> Aras.ITAF.Tests.TestCases.TestSamples.RunAs.RunAsTestUser
[Aras.Crt.IntegrationTests]:> Aras.ITAF.Tests.TestCases.TestSamples.SetupTest.SetupTest.SetupTestExample1
[Aras.Crt.IntegrationTests]:> Aras.ITAF.Tests.TestCases.TestSamples.SetupTest.SetupTest.SetupTestExample2
[Aras.Crt.IntegrationTests]:> Aras.ITAF.Tests.TestCases.TestSamples.WorkFlow.WorkFlow.WorkFlowMap
[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\project1\localRepo\Tests
[Aras.Crt.IntegrationTests]:   StopOnError: True
[Aras.Crt.IntegrationTests]:   BasePath: C:\ArasProjects\project1\localRepo\Tests\Output\Aras.ITAF.Tests
[Aras.Crt.IntegrationTests]:   ConfigurationFile: Aras.ITAF.Tests.dll.config
[Aras.Crt.IntegrationTests]:   AutoInPath: True
[Aras.Crt.IntegrationTests]:   ProcessMode: Default
[Aras.Crt.IntegrationTests]:   DomainName: 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: 7, Passed: 7, Failed: 0, Warnings: 0, Inconclusive: 0, Skipped: 0
[Aras.Crt.IntegrationTests]: Start time: 2023-03-30 15:17:57Z
[Aras.Crt.IntegrationTests]: End time: 2023-03-30 15:20:13Z
[Aras.Crt.IntegrationTests]: Duration: 135.372 seconds
[Aras.Crt.IntegrationTests]:
[Aras.Crt.IntegrationTests]: Results (html): saved as C:\ArasProjects\project1\localRepo\AutomatedProcedures\
Output\Logs\InstanceTestsStage\IntegrationTests\Results\Aras.ITAF.Tests.MUnitResults.xml

BUILD SUCCEEDED
Total time: 139.1 seconds.

BUILD SUCCEEDED - 0 non-fatal error(s), 1 warning(s)
Total time: 139.4 seconds.
SUCCESS!!!
PS C:\ArasProjects\project1\localRepo>
```



© 2023 Aras

Validating Build

To ensure that the new build is fine, execute ContinuousIntegration.ps1 again.

Try It ... Run Continuous Integration to Validate New Build

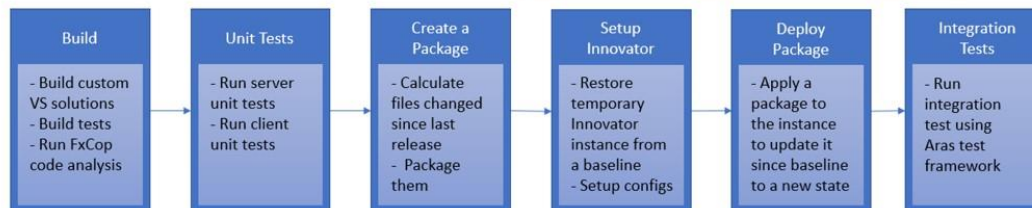
1. Right-click on the **Start** menu to open a new Windows PowerShell (Admin) window.
2. Access the local repository at C:\ArasProjects\project1\localRepo.
3. Run **ContinuousIntegration.ps1** as administrator.

Rebuilding Aras Innovator to Confirm New Configuration

- Rebuild Aras Innovator instance by executing the BuildAndDeploy script
- **Make sure to export all work BEFORE rebuild!!!**
 - Hint: Or create all commits in your local repo and switch to another branch and build new instance from that branch. Instances are created based on the branches and do not touch each other.

```

PS C:\Aras\Projects\project1\LocalRepo> .\BuildAndDeploy.ps1
Print Url of Installed Innovator:
URL of configured Innovator is: http://localhost:8080/ArasTraining
BUILD SUCCEEDED
Press any key to continue . . .
PS C:\Aras\Projects\project1\LocalRepo>
  
```



Rebuilding Aras Innovator to Confirm New Configuration

Once you added and committed the project files in the AML Package directory, you can rebuild the environment at any time using the **BuildAndDeploy.ps1** file. 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 are stored in the AML Packages folder in the development repository.

Try It ... Rebuild the Environment

1. Access the repository directory and make sure the current sprint branch (training) has been checked out.
2. Use Windows PowerShell (as administrator) to run the BuildAndDeploy.ps1 script.
3. 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?).

Note: To avoid the issue of losing changes you have not previously exported. Run ContinuousIntegration.ps1 first.

Summary

In this short introduction to Aras DevOps, you learned the basic guidelines for packaging items for Aras Innovator and how to use the packages within the Aras DevOps Framework SDE.

You also built a simple project based on a use case, and then exported the project into the customer repository after validating it locally through the Continuous Integration pipeline.

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