Aras Innovator 2023 Release

Package Import Export Utilities

Document #: *D-008119*

Last Modified: *12/8/2022*

Copyright Information

Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810

Phone: 978-806-9400
Fax: 978-794-9826

E-mail: Support@aras.com

Website: https://www.aras.com

Table of Contents

Send Us Your Comments

Aras Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for future revisions.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where and what level of detail?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, indicate the document title, and the chapter, section, and page number (if available).

You can send comments to us in the following ways:

Email:
TechDocs@aras.com
Subject: Aras Product Documentation

Or,

Postal service:
Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810
Attention: Aras Technical Documentation

If you would like a reply, provide your name, email address, address, and telephone number.

If you have usage issues with the software, visit https://www.aras.com/support/.

Document Conventions

The following table highlights the document conventions used in the document.

Convention	Description
Bold	This shows the names of menu items, dialog boxes, dialog box elements, and commands. Example: Click **OK**.
`Code`	Code examples appear in `courier` text. It may represent text you type or data you read.
`Yellow highlight`	Code with yellow highlight is used to draw attention to the code that is being indicated in the content.
`Yellow highlight with red text`	Red color text with yellow highlight is used to indicate the code parameter that needs to be changed or replaced.
Italics	Reference to other documents.
Note:	Notes contain additional useful information.
Warning	Warning contains important information. Pay special attention to information highlighted this way.
Successive menu choices	Successive menu choices may appear with a greater than sign (-->) between the items that you will select consecutively. Example: Navigate to **File** --> **Save** --> **OK**.

1 Overview

The main implementation method for some new functionality in Aras Innovator is the creation of a set of Items that encapsulate the desired functionality:

- Some Items represent business objects: ItemTypes and their instances
- Some support user interaction: Forms, etc.
- Some may implement business logic: Methods, etc.

Because all Aras Innovator Items are stored in the database, it creates numerous problems with identifying differences between different installations of Aras Innovator, keeping track of them, merging the differences, and upgrading to new releases. To implement a better mechanism for solving these problems, Aras Corporation has created a set of tools called **Package Import Export Utilities** for the import, export, and management of AML packages. The main concepts on which this set of tools is based are the following:

1. Each Item is represented as a separate AML file, which, in its turn, would allow:
    - The use of visual comparison between different versions of the same AML file.
    - The use of third-party visual merge tools for merging differences between different versions of the same AML file.
    - Keeping custom solutions or any set of related Items, including those that implement core Aras Innovator functionality, in the XML form in a file structure and keeping track of their changes.
2. A set of logically related Items is collected into a package.

Together these two concepts allow simplifying the process of importing and exporting Aras Innovator core, BRS, and custom solutions to and from the database. This allows administrators to keep track of modifications, merge differences between packages, and migrate these changes between databases.

The main goals of the Package Import Export Utilities set are the following:

- The ability to create and modify AML packages in the database.
- The ability to export some or all components of an AML package from a database to the file system in the form of a hierarchal set of AML files.
- The ability to import a hierarchy of AML files that represent a package to a database.
- If the package already exists in the database, the import process must provide an ability to automatically merge the differences between Items into the database from a corresponding imported AML.

2 Data Model

Conceptually, each package is a collection of Item IDs. Certain ItemTypes are to support package functionality. The following represents the common set of Items that defines an AML package:

- **PackageElement**: an ID number of an Item defined in the database.
- **PackageGroup**: a type of an ItemType (Method, List, etc.) that the Package Elements are added from. This ItemType also defines the name of the folder to which the Package Element is exported in the file structure.
- **PackageDefinition**: a package itself—a collection of package groups that make up this package and allow for the grouping of one package separate from another.
- **PackageDependsOn**: a RelationshipType that establishes dependencies between packages
- **PackageReferencedElement**: A relationship on the **PackageDependsOn** RelationshipType that defines what exact package elements from a *related* package the *source* package references. This could be useful when packages are exported (the **Export Referenced Items** checkbox in the Export tool; see the Export Tool section for more details).

Note: No Package Element may belong to more than one AML package. This is to prevent conflicts when importing the packages if these two elements are not identical in each package. The import would have no way of knowing which AML the correct AML was to apply otherwise.

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

- **Core Packages**: packages that are used to define the basic structure of every Aras Innovator database, regardless of what solutions are used in the database.
- **Solution Packages**: packages that define the elements that comprise the definition and functional rules of different BRS data models.

3 Using Package Import Export Utilities

3.1 AML Packages

The first step in understanding the use of the Package Import Export Utilities is to understand the structure of package AML files on disk and the corresponding manifest file.

3.1.1 The File Structure

The folder structure of a core package can be defined by careful use of the Package Definition name. For example, the fully qualified Package Definition name of the core **Dashboard** package is **com.aras.innovator.dashboard**. When exported, this package is exported to the following hierarchal structure:

Innovator/

Imports/

Com/

Aras/

Innovator/

Dashboard/

New core packages are treated as such and allow for the export of packages in this manner.

Non-core packages do not use the abovementioned rule for export, even though the solution packages have a fully qualified name. for example, such a name is not translated into a directory hierarchy of **com\aras\...** Instead, solution packages are always exported to the following three predefined folders:

Solutions/

PLM/

Import/

Project/

Import/

QP/

Import/

3.1.2 The Manifest File

A manifest file contains information about the following:

- Which packages can be processed by the utilities.
- What dependencies exist between packages.
- Where to find the package AML files.

The following is a manifest file example:

```
<imports>
   <package name="com.aras.innovator.solution.PLM" path="PLM\import" />
   <package name="com.aras.innovator.solution.QP"  path="QP\import" >
       <dependson name="com.aras.innovator.solution.PLM" />
   </package>
   <package name="com.aras.innovator.solution.Project"
path="Project\import">
       <dependson name="com.aras.innovator.solution.PLM" />
   </package>
</imports>
```

3.1.2.1 The package tag

The package tag can have the following attributes:

- name: a unique, fully qualified name of a package.
- path: a path to a directory that contains package AML files.

Note: A path to package AML files could be either absolute or relative. In the latter case, it is relative to the location of a manifest file.

For all non-core packages, the path attribute should include a path to the directory with the folders for different AML types. In the example given in The File Structure section, the manifest file for the **com.aras.innovator.solutions.PLM** package is in the **Solutions** folder, and the path to the PLM solution AML points to the **\PLM\Import** folder, where the **\ItemType**, **\Form**, and other folders are.

A package tag with a path relative to the manifest file location should be written as follows:

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

For core packages (Admin, Core, Dashboards, and Preferences), a path to package AML files is calculated based on the specific package name. A dot character (**.**) in the fully qualified name of a core package is replaced with a backslash character (****) when composing the file path based on these specific package names. In the example given in The File Structure section, the manifest file for the **com.aras.innovator.dashboards** package is in the **\Innovator\Imports** folder, thus the package tag should be written as follows:

```
<imports>
   <package name="com.aras.innovator.dashboards" path=".\" />
</imports>
```

3.1.2.2 The depenedson tag

The depenedson tag contains the information about packages on which the package defined in the package tag depends. This also populates the **Package Depends On** Relationship of the **Package Definition** Item in the database. This information is used for creating dependencies between packages in the database. If a package specified in the dependson tag is imported during the given import session, it must be loaded prior to the package that depends on it; otherwise, it is assumed that the package referenced by the dependson tag already exists in the target database.

Note: If it does not, the import might fail because an imported package might contain references to some Items from the dependson package.

3.2 Export Tool

With the Export tool, a user can export selected Package Elements to the file system as XML files with AML data. These Package Elements can be exported individually as a part of a Package Group or Package Definition.

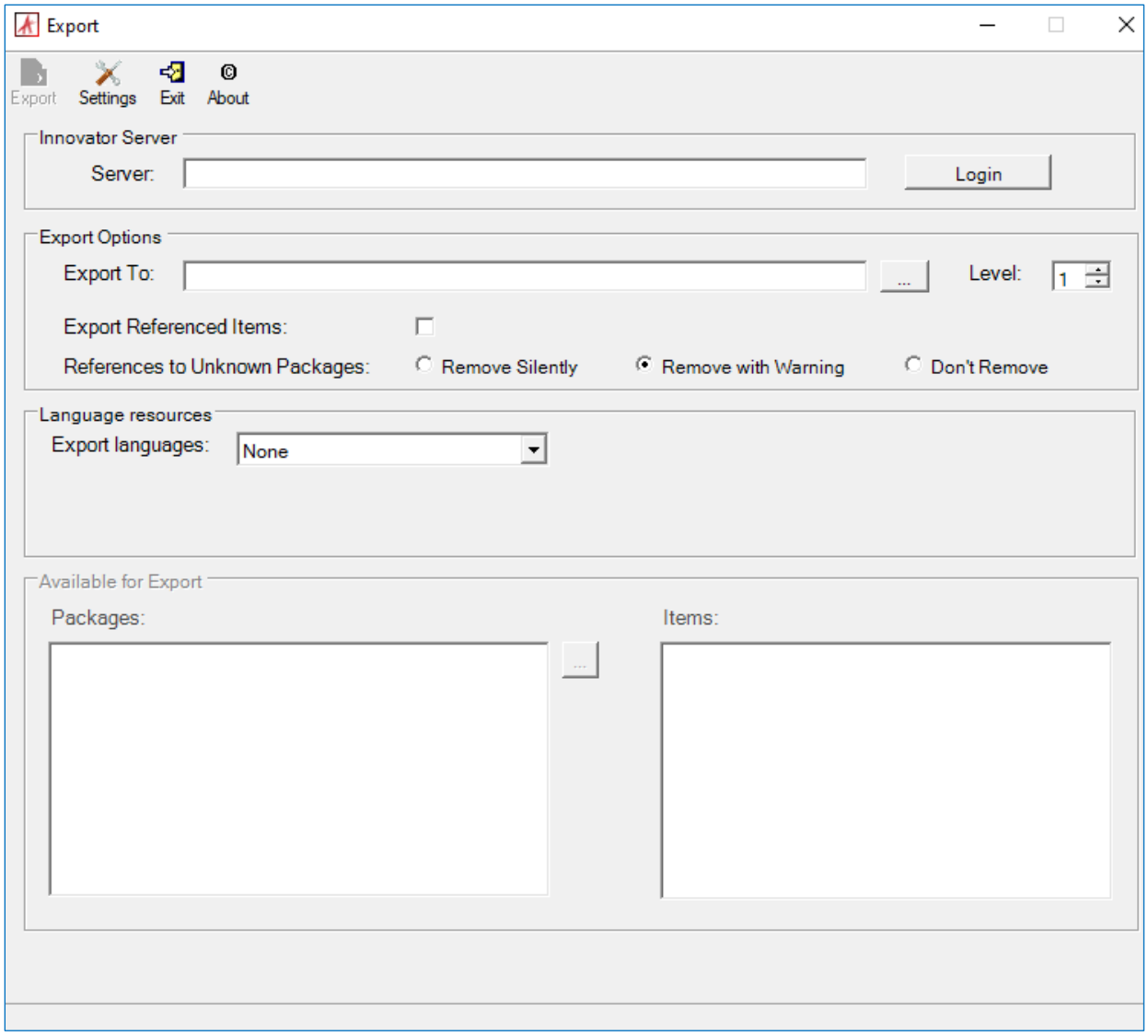

Figure 1.

To export Package Elements:

1. Enter a connection URL for a given Aras Innovator instance in the **Server** field, such as `http://localhost/InnovatorServer` and click **Login**.

If a given Aras Innovator server is configured for Client Certificate Authentication (CCA), the **Windows Security** dialog box appears.

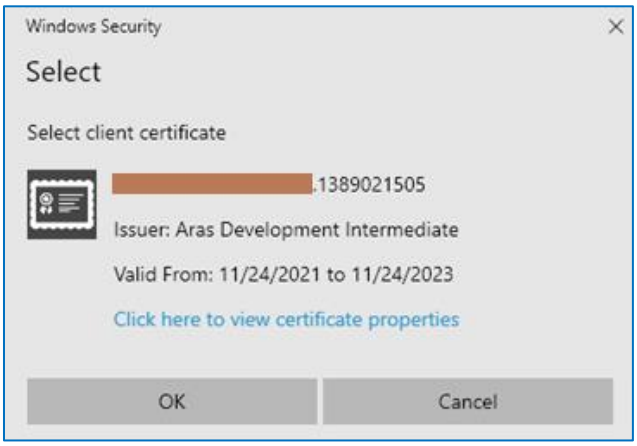

Figure 2.

2. Select a valid client certificate and click **OK**.

   The **Login to Aras Innovator** dialog box appears.

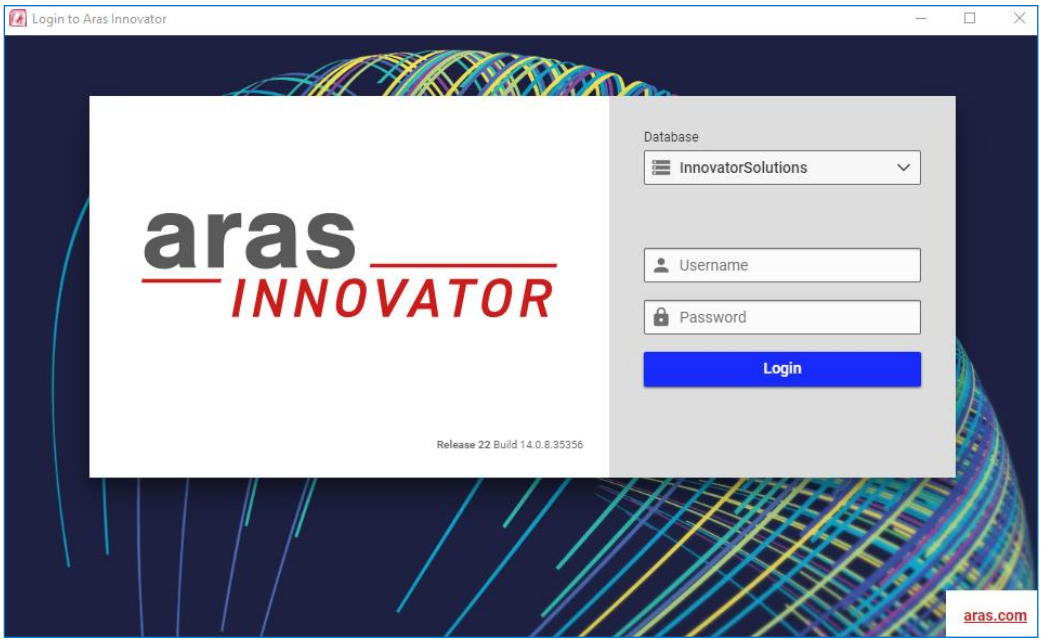

Figure 3.

3. Specify the connection information for the given Aras Innovator instance:
   - **Database**: a target Aras Innovator database.
   - **Username**: an Aras Innovator User under whose Identity the system will perform exporting. It is usually the root or admin Users when working in a non-production or upgrade environment.
   - **Password**: a password for the User given in the **Username** field.
4. Click **Login**. The Export tool should have successfully connected to the given Aras Innovator instance.

5. Fill in the Export tool fields:
    - **Export To**: a location in the file system where the AML packages will be exported.
    - **Levels**: a value for the `levels` attribute of the query in a limited number of ItemType queries not pre-defined by the tool.
    - **Export Referenced Items**: a flag when selected to export Items explicitly defined in the package definition as a referenced Item.
    - **References to Unknown Packages**: an exclusive option on what should be done with Unknown Packages.

      An Unknown Package is a package that is neither a core package nor a one on which an exported package depends. An exported package depends on another package when a dependency of the `PackageDependsOn` type exists between these packages.

      If an Item has references to Items in an Unknown Package, these references are normally removed during the export. For example, the **Quality Planning** Identity is used in the **New Part** Permission, and because the Quality Planning solution depends on the Product Engineering solutions exporting this relationship could create a circular reference.

      The options for Unknown Packages are the following:

- **Remove Silently**: references to Unknown Packages are removed. For example, the **New Part** Permission will not refer to the **Quality Planning** Identity, and this removal is not logged.
- **Remove with Warning**: references to Unknown Packages are removed, and a warning is given to allow the user to resolve this reference separately. For example, the **New Part** Permission will not refer to the **Quality Planning** Identity.
- **Don't Remove**: references to Unknown Packages are not removed. For example, the **New Part** Permission will not refer to the **Quality Planning** Identity. This may cause import errors if the **Quality Planning** Identity does not exist in the database during the package import.
    - **Export Languages**: an exclusive option on what should be done with language packs for the exported packages:
- **None**: the main language pack is exported, but additional language packs are not exported. For example, an original English text is exported, but German translations are not exported.
- **Items with translations**: the main and additional language packs are exported. For example, an original English text and German translations are exported.
- **Translations only**: the main language pack is not exported, but additional language packs are exported. For example, an original English text is not exported, but German translations are exported.

6. Choose what Package Elements should be exported in the Available for Export section. Click the **ellipsis** button next to the left panel to refresh the list of available elements for export.

   It is possible to export only a part of a package by expanding the package in the list of packages on the bottom left of the form and selecting a particular package group in the tree or items in the list of items on the bottom right of the form for export.

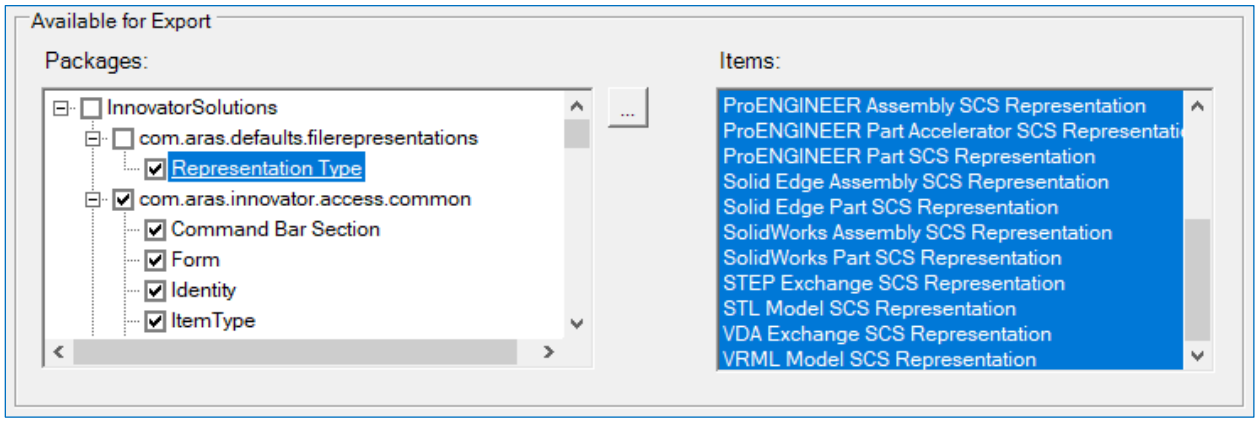

Figure 4.

7. Click the **Export** button to start the export.

   Once the export is finished, the **complete** message appears in the bottom left corner of the Export tool. The exported packages are available in the location specified in the **Export To** field.

3.3 Import Tool

With the Import tool, users can select predefined manifest files and import the corresponding Package Elements from XML files with AML data into a database.

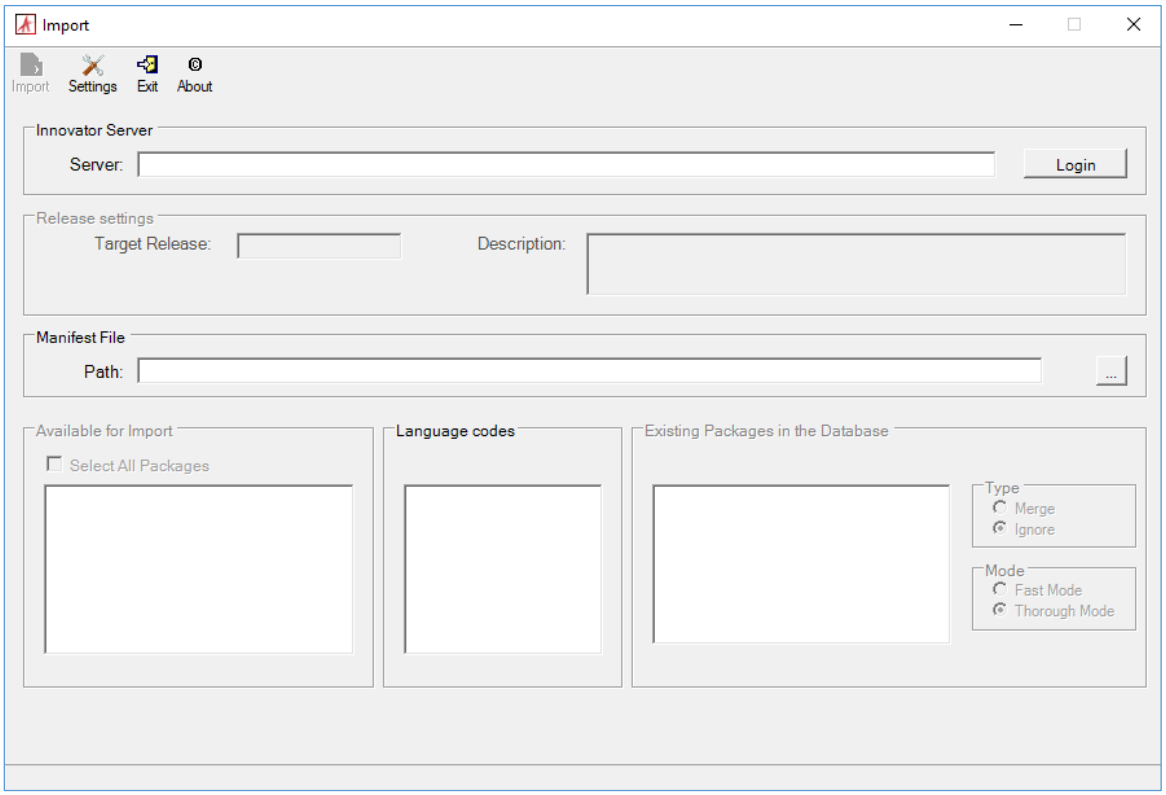

Figure 5.

To import Package Elements:

8. Enter a connection URL for a given Aras Innovator instance in the **Server** field, such as `http://localhost/InnovatorServer` and click **Login**.

   If a given Aras Innovator server is configured for Client Certificate Authentication (CCA), the **Windows Security** dialog box appears.

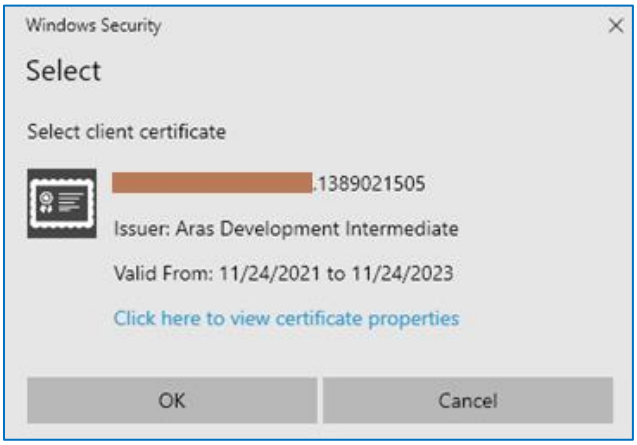

Figure 6.

9. Select a valid client certificate and click **OK**.

   The **Login to Aras Innovator** dialog box appears.

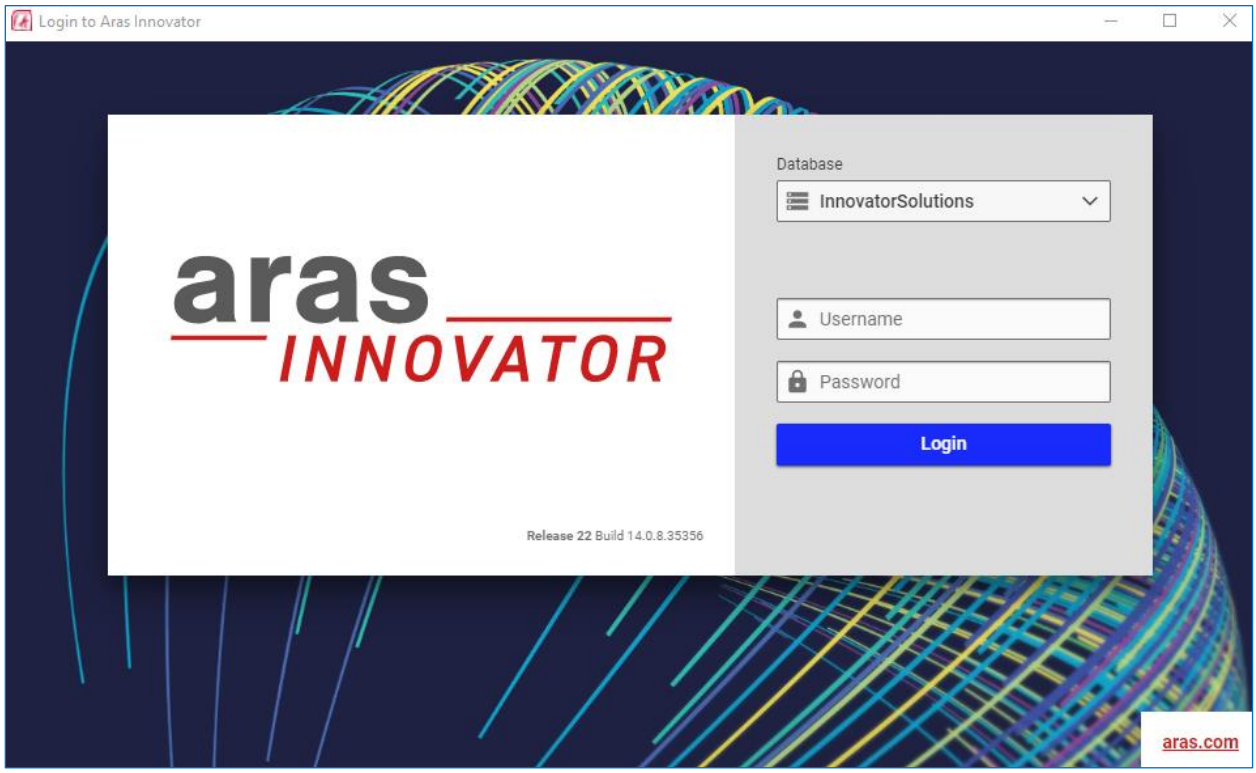

Figure 7.

10. Specify the connection information for the given Aras Innovator instance:
    - **Database**: a target Aras Innovator database. The default URL is **http://localhost/InnovatorServer**.
    - **Username**: an Aras Innovator User under whose Identity the system will perform importing. It is usually the root or admin Users when working in a non-production or upgrade environment.
    - **Password**: a password for the User given in the **Username** field.
11. Click **Login**. The Export tool should have successfully connected to the given Aras Innovator instance.
12. Fill in the Import tool fields:
    - **Target Release**: a target release number of the database after the import is completed. When migrating packages, this is the current version number of the database, for example, 14-to-14 import. When upgrading a database, this is the upgrade target version of the database, for example, 9.4.0-to-14 import.

      This information is also used by Aras Innovator classes to obtain information for a user about database upgrades applied to the database.
    - **Description**: A brief description of the import purpose.
    - **Manifest File Path**: a manifest file that contains information about what packages and from where on a disk should be imported into the database.
    - **Available for Import**: packages from the manifest file to be imported to the database.

- **Type**: to define how conflicts with existing packages in the database should be resolved:
  - **Ignore**: to skip imported Items if they already exist in the database.
  - **Merge**: to update package Items in the database with the new AML.

    If an Item action specified in AML is not `add`, it is always used as is.

    If an Item action specified in AML is `add`, it is replaced by `edit` if a given Item with the ID already exists in the database.
- **Mode**: to determine the level of error checking that is performed before an AML is imported to the database.
  - **Fast**: no additional verifications of the imported AML are done during the import process.
  - **Thorough**: during the import, additional checks about the AML's dependent Items are checked for existence in the database before applying. For example, making sure all properties exist with the correct ID before attempting to apply an ItemType.

13. Click **Import** to start the installation.

    Once the import is successfully finished, the **complete** message appears in the bottom left corner of the Import tool. The imported packages are available in the Aras Innovator instance.

It is recommended that AML import files always contain the `add` Item action for every imported Item unless it is specifically required otherwise; for example, an Item with a particular ID must be removed from the database. Another important thing to understand is how the import processes versionable Items. Package Elements always contains a `config_id` of a versionable Item. When a versionable Item is imported, the tool tries first to find its `config_id` and then uses the ID in the Package Element. Correspondingly, exporting a versionable Item always writes its `config_id` as ID in the resulting AML file; see the Export Tool section.

3.4 Console Upgrade Tool

The Console Upgrade Tool is a command-line version of both the Export and Import Tools described above. The command-line parameters can be obtained by typing `'/?'` as a command-line parameter.

3.4.1 Required Parameters

- `server=url`: a server URL, for example, `server=http://localhost/InnovatorServer`.
- `database=db`: a name of a database, for example, `database=dbWithoutSolution`.
- `login=uname`: a user's login, for example, `login=root`. The login must have root or admin privileges.
- `password=pw`: a user's password, for example, `password=xxxx`.
- `release=rel`: release's name used by import only, for example, `release=rel11.0`.
- `mfFile=path`: an explicit path to a **.mf** file. It is not required for export. The default behavior is to export all.

3.4.2 Optional Parameters

- `import`: `import` or `export`. It is `export` by default.
- `merge`: for import only. It is `non-merge` by default.

- `fastmode`: for import only. By default, it uses the `thorough` mode that provides more thorough verification of the applied AML.
- `verbose`: for import and export. It is `non-verbose` by default.
- `dir=d1`:
  - Export: an output directory.
  - Import: a location of a manifest file.

  The specified path must exist. If this parameter is not specified at all, a user is prompted for the path.
- `log=logpath`: a full path to the log file. If a specified file already exists, it is appended.
- `description=desc`: a release description or import only, for example, `description="SolutionsUpgrade"`.
- `vlog`: save the resulting log file on the vault (don't save if the argument wasn't specified).
- `level=n`: a request attribute `level` is used for non-dictionary ItemTypes for export only. By default, it is `level=1`.

3.4.3 CCA Arguments

The CLI of the Console Upgrade Tool has been extended with the following arguments:

- **`crtloc`**: a name of the certificate's store location (CurrentUser, LocalMachine) used for Client Certificate Authentication; default: CurrentUser.
- **`crtstore`**: a name of the certificate's store used for Client Certificate Authentication; default: My.
- **`crtftype`**: a type of a search criteria that will be used to find a client certificate in the specified store; default: FindBySubjectName. Here is the list of available values: https://docs.microsoft.com/dotnet/api/system.security.cryptography.x509certificates.x509findtype?view=net-6.0
- **`crtfvalue`**: a value of a search criteria that will be used to find a client certificate in the specified store.
- **`crtvalid`**: a parameter that determines whether to accept only valid certificates (true) or any certificate (false); default value is true.

Also, users can specify data about the client certificate in the 'ConsoleUpgrade.exe.config' file. An example of the configuration is below:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <sectionGroup name="Aras">
      <sectionGroup name="Net">
        <section name="RequestProvider"
type="Aras.Net.Configuration.RequestProviderConfigurationSection, IOM"/>
      </sectionGroup>
    </sectionGroup>
  </configSections>
  <Aras>
    <Net>
      <RequestProvider>
        <providers>
          <provider uriPattern=".*">
            <ClientCertificate>
              <certificate sourceType="CertificateStore" storeLocation="CurrentUser"
storeName="My" findType="FindByThumbprint"
findValue="71f942b0710faa873d1d3de2a9815b3321604d5a"/>
            </ClientCertificate>
          </provider>
```

```
        </providers>
      </RequestProvider>
    </Net>
  </Aras>
</configuration>
```

3.5 Package Definition Tool

The Package Definition Tool allows creating an instance of the Package Definition in the database. This is a temporary utility that is created only for the situations when Aras Innovator has a set of items that makes up a Package Definition, but these items are not included as part of a Package Definition. Generally, this only occurs when the database was upgraded from a version of Aras Innovator that pre-dated the corresponding use of Package Definitions for this solution.

Note: The Package Definition Tool does NOT import anything into the database. It creates a Package Definition in the database that later can be used for managing the Package Elements.

3.5.1 The Package Definition Tool GUI

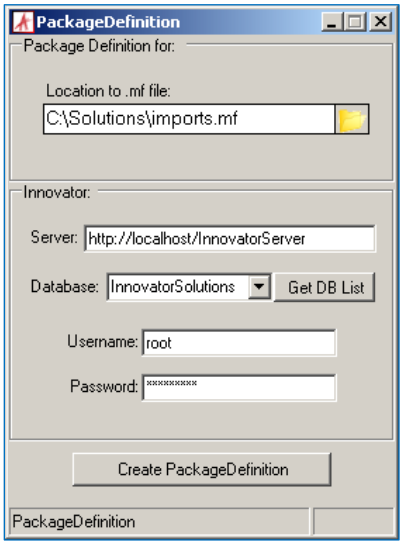

Figure 8.

1. The first step in using this utility is specifying a manifest file that contains information about what packages and from where on disk should be defined in the database.
2. The next step is to establish the connection parameters:
   - **Server**: The URL used to login to Aras Innovator. A default install uses a URL like http://localhost/InnovatorServer
   - **Database**: The database you wish to define the package in. This field is populated by selecting the 'Get DB List' button, after filling out the Server URL field.
   - **Username**: The login of the user that is used to log into Aras Innovator. This is usually the root or admin logins when working in a non-production or upgrade environment.
   - **Password**: This is the password for the login filled out in the Username field.

   After all of this information is filled out, click **Create Package Definition** to begin.

3.5.2 The Package Definition Tool Command Line

Alternately, the Package definition tool can be executed from the command line. The command line parameters can be obtained by typing '/?' as the command line parameter.

Usage:

```
PackageDefinition.exe {server url} {user} {password} {db} {path to manifest
file}
 server url              server's URL (e.g. http://localhost/InnovatorServer)
 user                    user's login
                         NOTE: login must have root or admin privileges
 password                user's password
 database                database's name
 path to manifest file   location of the manifest file
                         NOTE: the specified path must exist
```

EXAMPLE:

```
C:\PackageImportExportUtility\PackageDefinition\PackageDefinition.exe
http://localhost/InnovatorServer root innovator InnovatorSolutions
C:\Solutions\core_imports.mf
```

3.6 Creating a Package Definition from the Aras Innovator UI

Package Definitions can also be created for new solutions using administrator features in the Aras Innovator user interface. These packages should be created with a specific data model in mind, and you should be careful to review your package for any referenced items that may not be in the next database you import to. Lists, for instance, are commonly forgotten when creating a package definition, but causes errors when trying to import ItemTypes that reference them.

To create a package definition, you must be an administrator in Aras Innovator.

1. From the main search grid right click the Item you wish to add to a package. A menu like the following appears:

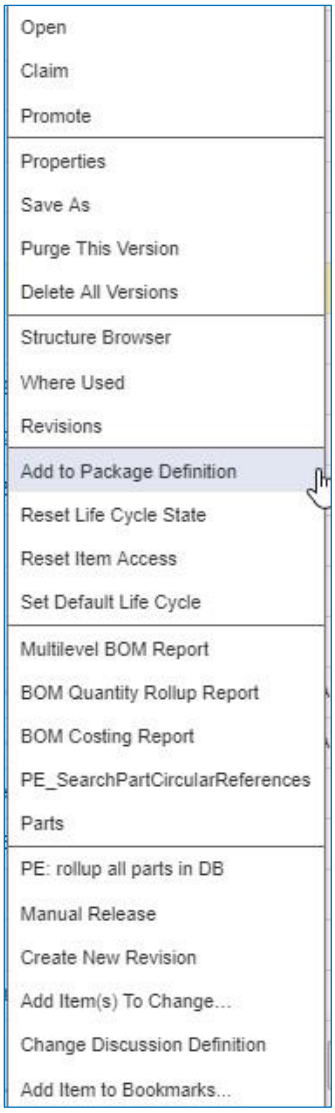

Figure 9.

2. Click **Add to Package Definition**. The following dialog appears.

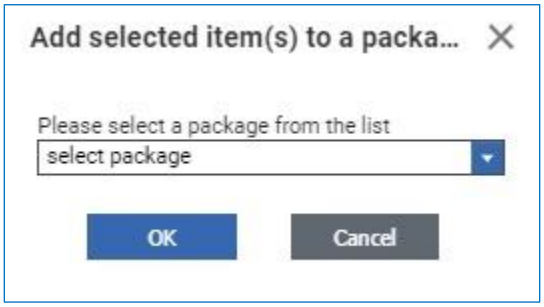

Figure 10.

3. From the dialog, ether select an existing package or **create new**.

   If you select an existing package, the item is added to the existing package, and you are done.

4. If you select **create new**, you are prompted to define the new package.

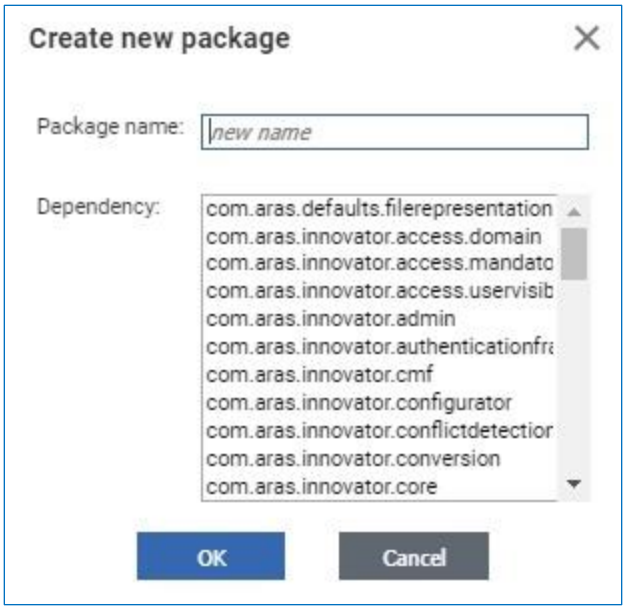

Figure 11.

- Package name should be alphanumeric without spaces, as a best practice. The package name is used to create file folder names on export.
- Dependency should be the packages that are required to exist in the database before the defined package can be imported.

After you have created your package, you may review it and other existing package definitions by selecting Administration\Package Definitions in the TOC.

3.6.1 What to include in a Solutions AML Package

There is no fixed list for what Items to include in a solutions AML package, because the list would change based on the version of Aras Innovator or solution you are working with. However, there is a basic list of the core ItemTypes that make up the basic metadata of a database. This is NOT a definitive list but should act as a helpful guideline when creating your packages.

- Actions
- CommandBar… (all items)
- Conversion Rules
- Derived Relationship Fmilies
- E-Mail Messages
- File Types
- Forms
- Grids
- Identities
- ItemTypes (Exclude ItemTypes with is_relationship=1)
- Life Cycle Maps
- Lists (Exclude Lists associated with PolySources like ‘Change Controlled Item’ and ‘Deliverable’)
- Mac Policies
- Methods
- Permissions
- Presentation Configurations
- Query Definitions
- RelationshipTypes
- Reports
- Sequences
- SQLs
- User Messages
- Variables
- Workflow Maps
- xClassification Trees
- XML Schemas
- XML Schema Elements
- xProperty Definitions