



## OpenEdge DevOps Framework



# Copyright

---

Visit the following page online to see Progress Software Corporation's current Product Documentation Copyright Notice/Trademark Legend: <https://www.progress.com/legal/documentation-copyright>.

**April 2026**

**Product version:** Progress OpenEdge DevOps Framework 2.4

**Updated:** 2026/04/02



# Table of Contents

<b>Preface.....</b>	<b>7</b>
<b>Part I: Introduction.....</b>	<b>9</b>
Learn About the OpenEdge DevOps Framework.....	9
<b>Part II: Gradle plugins.....</b>	<b>11</b>
Prerequisites.....	12
Learn the basics.....	15
Gradle.....	15
Understand build process of an ABL project.....	15
Manage dependencies.....	16
Compile source code.....	16
Run unit test.....	17
Run static code analysis.....	17
Generate API documentation.....	18
Package and publish.....	18
Plugins.....	19
ABL base plugin.....	20
Task types.....	20
Task types for Progress Application Server (PAS).....	44
Dependency management.....	63
Global configurations using the contributed extension .....	64
Transform.....	67
ABL plugin.....	67
Propath file.....	68
Build config file.....	68
Tasks.....	74
Dependency Management.....	76
Build packages for PAS for OpenEdge application.....	79
Migrate from Apache Ant.....	80



---

# Preface

---

## Purpose

This manual provides information about OpenEdge DevOps framework for ABL applications. The OpenEdge DevOps framework is designed to implement an efficient Continuous Integration (CI) pipeline to handle compilation, repository integration, testing, and packaging.

## Audience

This document is intended for OpenEdge DevOps Engineers, and OpenEdge Developers.

## Organization

- [Learn About the OpenEdge DevOps Framework](#) on page 9

This section provides an introduction to OpenEdge DevOps Framework.

- [Gradle](#) on page 15

This section covers the prerequisites for using the OpenEdge DevOps Framework and some basic concepts of Gradle.

## Documentation conventions

See [Documentation Conventions](#) for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content library.

## Purpose

This manual provides information about OpenEdge DevOps framework for ABL applications. The OpenEdge DevOps framework is designed to implement an efficient Continuous Integration (CI) pipeline to handle compilation, repository integration, testing, and packaging.

## Audience

This document is intended for OpenEdge DevOps Engineers, and OpenEdge Developers.

## Organization

- [Learn About the OpenEdge DevOps Framework](#) on page 9

This section provides an introduction to OpenEdge DevOps Framework.

- [Gradle](#) on page 15

This section covers the prerequisites for using the OpenEdge DevOps Framework and some basic concepts of Gradle.

## Documentation conventions

See [Documentation Conventions](#) for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content library.



---

# Introduction

---

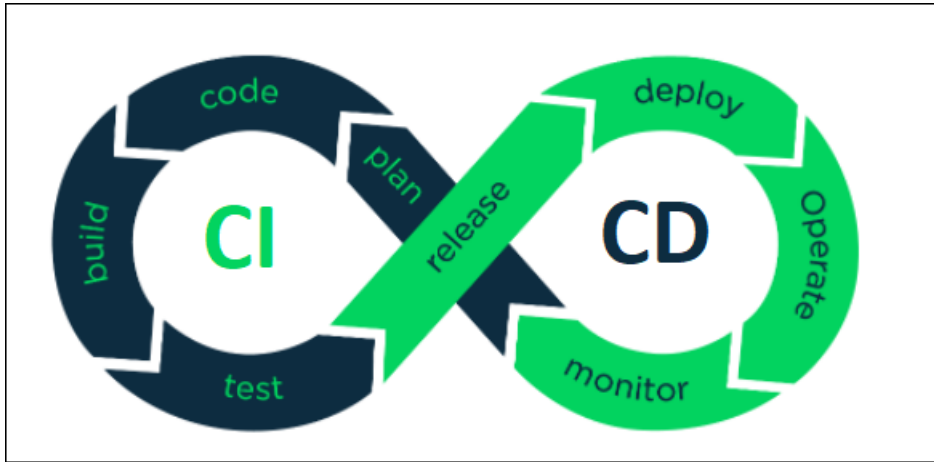
For details, see the following topics:

- [Learn About the OpenEdge DevOps Framework](#)

## Learn About the OpenEdge DevOps Framework

Read the [End User License Agreement](#).

Mission critical business applications go through many cycles of incremental changes over the course of their lifespan, complete with code changes, compilation, builds, and validation. Often these cycles are repeated multiple times a day, before a release is made available to users. Continuous integration (CI) is a practice that focuses on optimizing the process of generating these incremental builds and validating them. In the illustration that follows, CI represents DevOps and describes a typical software development and delivery process.



For ABL applications, the OpenEdge DevOps Framework is designed to help with implementing an efficient CI pipeline that handles compilation, repository integration, testing, and packaging. It also provides the convenience of sharing the CI pipeline configuration between the development and production build processes.

The OpenEdge DevOps Framework comprises a set of plugins designed to address the requirements of two types of users:

1. Users who are new to the CI process and want a simple way to set up their pipeline.
2. Advanced DevOps engineers with a complex CI process and additional flexibility needs.

The OpenEdge DevOps Framework provides the convenience of sharing the CI pipeline configuration between the development and production build processes.

The following sections cover prerequisites for using the OpenEdge DevOps Framework, basic concepts of Gradle, understanding build process, managing dependencies, and details of the ABL Base plugin and the ABL plugin.



# Gradle plugins

---

For details, see the following topics:

- [Prerequisites](#)
- [Learn the basics](#)
- [Plugins](#)
- [Build packages for PAS for OpenEdge application](#)
- [Migrate from Apache Ant](#)

## Prerequisites

To use the OpenEdge DevOps Framework (OEDF) Gradle plugin, ensure that you have the following:

- Certified Progress OpenEdge® 12.2 or later.
- OpenEdge ABL installed (version 12.2 or later).
- Gradle 9.3.1.
- JDK 17 or later.

---

**Note:** Running OEDF 2.4 with Gradle 9.3.1 on JDK 11 is not supported because Gradle 9.x.y requires JDK 17 or later.

---

- OEDF is supported on AIX, Linux, and Windows, which is consistent with the platforms supported by OpenEdge 12.2.x and later versions. Oracle ended Java support for Solaris with JDK 15. Since OEDF 2.4 requires JDK 17 or later, it does not run on Solaris. For Solaris environments, you can continue using OEDF 2.3.
- [PCT](#) (Progress Compile Tool):
  - By default, the plugin uses the PCT version available in your OpenEdge installation (for example, the `DLC` directory), or the default PCT available in your environment.
  - Optionally, you can use a different PCT version by setting the `PCT_VERSION` property using:
    - A system property
    - A Gradle property
    - An environment variable
- An ABL project
- A basic understanding of Gradle

### Set up the environment

OpenEdge provides the `progradle.bat` script to simplify the OpenEdge DevOps Framework (OEDF) setup on your machine. This script prepares the environment required to run the Gradle builds that use the OEDF plugins.

When you execute the `progradle.bat` script, the following actions are performed in the background:

- The specified Gradle version is automatically installed when you run the script for the first time.

To determine the supported Gradle version for your OpenEdge and OEDF versions, use the following **OpenEdge + OEDF → Gradle compatibility** matrix, and then update the `distributionUrl` property in the `%DLC%\gradle\wrapper-scripts\gradle\wrapper\gradle-wrapper.properties` file.

#### OpenEdge + OEDF → Gradle compatibility

OpenEdge version	OEDF 1.x	OEDF 2.0-2.2	OEDF 2.3.0	OEDF 2.4.x
OpenEdge 12.2	Gradle 5.6.x	Gradle 7.3.3	Gradle 7.6.x or 8.2.x	Gradle 9.3.x (requires JDK 17 for Gradle run-time)
OpenEdge 12.8	Gradle 5.6.x	Gradle 7.3.3	Gradle 7.6.x or 8.2.x	Gradle 9.3.x
OpenEdge 13.0	Gradle 5.6.x	Gradle 7.3.3	Gradle 7.6.x or 8.2.x	Gradle 9.3.x

- The OpenEdge JDK is automatically configured for running the Gradle build scripts.

As an alternative, you can install and configure the supported versions of Gradle and Java separately. For more information, see the [OpenEdge Platform and Product Availability Guide](#) for each OpenEdge release.

OEDF 2.4 uses Gradle 9.3.1. However, ProGradle continues to use the Gradle Wrapper configuration located at:

```
%DLC%\gradle\wrapper-scripts\gradle\wrapper\gradle-wrapper.properties
```

By default, this Gradle Wrapper configuration file references Gradle 8.2.1:

```
distributionUrl=https\://services.gradle.org/distributions/gradle-8.2.1-bin.zip
```

#### Why this update is required

OEDF 2.4 uses Gradle 9.3.1, and ProGradle must align with the same Gradle version. ProGradle reads the Gradle version directly from the `gradle-wrapper.properties` file. If this file continues to reference 8.2.1, ProGradle will use that version, causing a Gradle version mismatch during builds.

To update the Gradle Wrapper configuration manually, perform the following steps:

---

**Note:** Gradle 9.3.1 requires JDK 17 or later to run the Gradle Daemon. When running Gradle 9.3.1 with OpenEdge 12.2.x, ensure that JDK 17 or later is used as the Gradle run-time. This upgrade prevents the build from failing to start due to an unsupported Java run-time.

---

1. In a text editor, open the wrapper configuration file from  
%DLC%\gradle\wrapper-scripts\gradle\wrapper\gradle-wrapper.properties.
2. Locate the current `distributionUrl` value and replace it with the updated value.

**Current value:**

```
distributionUrl=https\://services.gradle.org/distributions/gradle-8.2.1-bin.zip
```

**Updated value:**

```
distributionUrl=https\://services.gradle.org/distributions/gradle-9.3.1-bin.zip
```

This change ensures that ProGradle downloads and uses Gradle 9.3.1.

3. Verify the Gradle version used by ProGradle by running the following command from `proenv`:

```
"C:\Progress\OpenEdge128\bin\progradle.bat" -v
```

4. Run a sample task with logging enabled:

```
"C:\Progress\OpenEdge128\bin\progradle.bat" --info helloProGradle
```

## Configure a property

OpenEdge DevOps Framework (OEDF) provides options to configure project builds by setting properties. It supports numerous properties depending on the project requirements. For example, when you set the `PCT_VERSION` property, it instructs OEDF to use `PCT` from Apache Maven instead of using it from the OpenEdge installation path.

You can configure a property in any of the following ways:

1. Set as a system property by running the following CLI command:

```
-DPCT_VERSION=<PCT-version>
```

2. Set as a Gradle property.

For example, set `PCT_VERSION=<PCT-version>` in the `gradle.properties` file.

3. Set as an environment variable.

The preferred order of precedence for this property (from highest to lowest) is 1 > 2 > 3.

# Learn the basics

This topic provides a basic understanding of Gradle and describes the typical build process followed for an ABL project.

## Gradle

[Gradle](#) is an open-source project automation tool, which is an evolution of the concepts used in Apache Ant and Apache Maven. Instead of traditional XML, Gradle uses a domain-specific language based on Groovy for project configuration. Gradle intelligently determines which parts of a build tree are up to date, and executes only those parts that are needed. This process allows builds to be smarter, faster, and more efficient because it eliminates redundancies in the build phase.

The OpenEdge DevOps Framework provides Gradle plugins to evolve the way ABL applications are built. Some of the Gradle terminologies and concepts that are used frequently across this guide are covered in the following Basic Concepts topic. For more information about Gradle, see the [Gradle documentation](#).

### Basic concepts

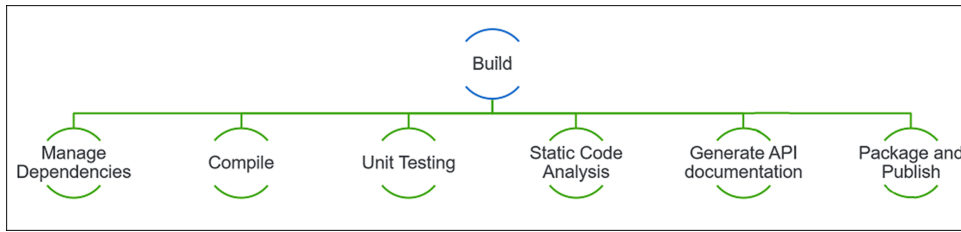
The following basic Gradle terminologies and concepts are helpful when using the OpenEdge DevOps Framework:

- **Plugin**—Gradle plugins add task objects and conventions to your Gradle environment.  
They are the primary method of extending capabilities using Gradle.
- **Task**—A Gradle task is the smallest piece of work for a build. For example, compiling classes or generating ABL doc.  
It is the building block of Gradle functionality.
- **Task type**—Defines the blueprint of a task.  
Some tasks may require configurations like input or output parameters to provide certain capability. The blueprint for these parameters is defined by the task type for the piece of work that can be set while creating a task.
- **Task dependencies**—Gradle tasks can depend on other tasks.
- **Extension**—Plugins can add Extension that allows you to configure several settings and properties for the plugins.  
Task types use these settings and properties as global configurations to set some of their properties.
- **Groovy DSL**—You can write Gradle build scripts using a [Groovy](#) or [Kotlin](#) DSL.

For more information about these and other Gradle components, see [Gradle documentation](#).

## Understand build process of an ABL project

Building an ABL project involves various steps such as, managing project dependencies, compiling ABL files, performing unit tests, generating ABL API documentation, packaging the artifacts, and publishing them to an artifact repository.



You should understand these steps in depth before writing a build script.

This topic is intended to familiarize you with the steps and requirements of building an ABL project.

## Manage dependencies

An ABL application can have various dependencies like library files containing ABL code, databases, application packages (for example, a `.war` files), resource files (images, icons, configurations), and many more. Thus, managing project dependencies is an important part of building an ABL project.

OpenEdge client or ABL Virtual Machine (AVM), which is the platform for compiling and running ABL code, uses **PROPATH** to search and locate ABL files and resource files. The following section describes all the dependencies that a **PROPATH** can manage.

### PROPATH specific dependencies

AVM can search and locate files based on following types of values in **PROPATH**:

- Path to a directory containing ABL code and resources.
- Path to a `Procedure Library` file containing only `r-code`.

### Procedure Library (PL) dependencies in PROPATH

Procedure Library (PL files) is the most common way to bundle ABL code as library. PL files can bundle `r-codes` and potentially any other source files (ABL files such as procedure files, class files, and include files) or resource files.

Because, AVM cannot recognize PL files with content other than `r-code` in **PROPATH**, you must handle PL file dependencies in the following ways:

- PL file that contain only `r-code`—Directly add to the **PROPATH**.
- PL file containing other source files (such as ABL sources, image files, config files, and other files)—First extract, and then add the path of the extracted directory to the **PROPATH**.

---

**Note:** The dependencies pertaining to database artifacts (such as schema file, structure file, backup file, and so on) and application packages (such as WAR files, OEAR files, PAAR files, and so on) are not yet supported.

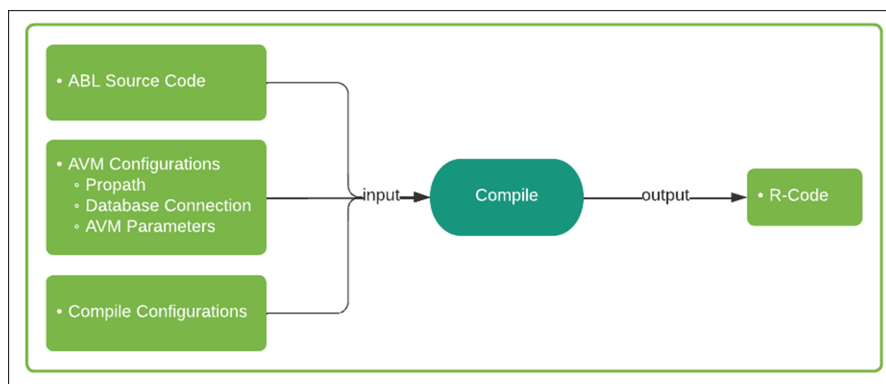
---

## Compile source code

Compiling source code refers to converting ABL source codes into the binary `r-code`, which is then executed inside the ABL Virtual Machine (AVM).

To compile an ABL code, you require an ABL source code and an AVM with compiler. The following illustration typically represents the input and output parameters for a compile operation:





## ABL source code

ABL files such as procedure files and class files that need to be compiled.

## AVM configurations

AVM configuration consists of:

1. Propath—The paths where the AVM looks for resolving various project dependencies. See [Manage dependencies](#) on page 16 for more details.
2. Database connection—For projects that require data from a database, you need to provide the database connection details.

In a CI environment, it is good to use a test database. Use single user mode ( -1 ) for connection details to overcome the overhead of starting a database server.

3. AVM parameters—The startup parameters as supported by AVM.

## Compile configurations

The `COMPILE` statement provides various options, such as `OPTIONS` (strict options), `LISTING`, and so on, which can be configured while compiling. For more information, see [COMPILE statement](#) in the *ABL Reference*.

## Run unit test

Running unit tests eliminates bugs at the *code level* or *unit level*. A unit (for example, a program module) is the smallest entity which can exist independently.

Unit testing verifies whether the smallest entity functions correctly when isolated from the rest of the code to ensure application features work at the component level.

OpenEdge provides the **ABL Unit testing framework** to unit test ABL programs. To run unit tests, you require sources (test files) and AVM. See [AVM configurations](#) to configure AVM for propath, database connection, and AVM parameters.

## Run static code analysis

Progress provides the following code analysis tools:

- Code Analyser for ABL (CABL)
- Sigrid

## Code Analyser for ABL (CABL)

Code Analyzer for ABL (CABL) is a third party plugin bundled with Progress Developer Studio for OpenEdge. It helps in analyzing and measuring the quality of your code and highlights any problems.

It also provides recommendations on how to fix the problem. This way, it helps to achieve coding best practices and improve product performance.

For more information, see "Introduction to Code Analyzer for ABL" in the *OpenEdge DevOps Framework* guide.

---

**Note:** Code Analyzer for ABL is currently not supported with OpenEdge DevOps Framework Gradle plugins.

---

## Sigrid

Progress professional services has partnered with Sigrid, a comprehensive software assurance platform that provides insights to both business and technical stakeholders on multiple software quality aspects, to measure, evaluate, and monitor the health of your entire software application at every stage of its life cycle. It exposes hidden risks and opportunities in your source code, provides continuous insights and recommendations on the performance of your software application.

For more information about Sigrid, contact the

<https://www.progress.com/services/contact?s=OpenEdge&ss=consulting-outsourcing>.

## Generate API documentation

You can generate API documentation (ABLDoc) in the default HTML format from ABL source code to refer to your code outside OpenEdge.

You can generate the ABLDoc documentation for the following source codes:

- Class (.cls) files.
- Procedure (.p) files.
- Include (.i) files.

---

**Note:** Generating ABLDoc is not natively supported by OpenEdge DevOps Framework Gradle plugins. You can alternatively use the ABLDoc ant task in your Gradle script.

---

## Package and publish

Packaging refers to creating and bundling the build output of an ABL application. How you package and potentially publish your ABL project depends on what type of project it is. For example, libraries, applications, and web applications (PAS for OpenEdge applications) have different requirements and produce different set of artifacts.

A common package step could be creating a Procedure Library (PL) artifact by packaging the compiled `r-code` files.

---

**Note:** You can write a custom task using the `PL` task type to create a PL artifact.

---

Server project (PAS for OpenEdge application) require creating `OEAR`, `OEDS`, `WAR`, or `PAAR` files.

---

**Note:** Tasks to create a server project package artifact is not yet supported in OEDF.

---

The packaged artifacts can then be published to an artifact repository using one of the Gradle's publishing plugins:

- [Maven publish plugin](#) .
- [Ivy publish plugin](#).

## Plugins

Plugins are the primary way of providing all the useful features in Gradle. They add various capabilities (such as task types to compile code) and conventions (such as creation of default tasks to build a project, depending on the project structure).

You can use a vast number of plugins from the Gradle community depending on the build needs such as:

- The [base plugin](#) that provides some tasks and conventions that are common to most builds.
- The [maven publish plugin](#) that provides the ability to publish build artifacts to an Apache Maven repository, and many more.

The OpenEdge DevOps Framework provides Gradle plugins for ABL projects. When creating Gradle plugins, you should separate the capabilities from convention because conventions can be opinionated (by having predefined tasks and defaults) and might expect a particular project structure (such as a Progress Developer Studio for OpenEdge project) whereas in real time scenario, projects might differ for some customers. Hence, the plugins provided by the OpenEdge DevOps Framework are designed to be flexible enough to cater to all the customer needs.

Currently, there are two ABL gradle plugins available:

- ABL base plugin ([progress.openedge.abl.base](#)).
- ABL plugin ([progress.openedge.abl](#)).

### ABL base plugin ([progress.openedge.abl-base](#))

The ABL base plugin defines various capabilities, under task types and extensions. Task types define individual task capabilities like compile ABL code, run ABL unit tests, and other tasks. Extensions are global configurations available to all the task types.

The ABL base plugin is intended for users who have ABL source files that are not structured as Progress Developer Studio projects. Users can create and define their tasks for various steps required to build an ABL project by using the task types; for example, using `ABLCompile` task type to create a task for compiling ABL code. For more information, see [ABL Base Plugin](#).

### ABL plugin ([progress.openedge.abl](#))

The ABL plugin depends on the Progress Developer Studio project structure and is recommended for the Progress Developer Studio users. This plugin, by default, provides various predefined tasks required to build an ABL project (such as task to compile ABL sources), based on the project's propath and properties so that you can build your project with minimum effort and enforce best build practices.

These predefined tasks and the necessary parameters required for these tasks (for example, compiling ABL files) can be configured in the `build.config` file that is part of each Developer Studio project. This way you do not have to maintain separate configuration for development environments and build environments.

Users who do not have a Progress Developer Studio for OpenEdge project but still prefer using predefined tasks can use the ABL plugin, provided they add and configure the required `build.config` and `.propath` files for their source artifacts. For more information about how to use and configure this plugin, see [ABL Plugin](#).

## ABL base plugin

The ABL base plugin provides various capabilities and features required to run or build ABL applications, such as running ABL procedure, compiling ABL sources, running ABL unit tests, creating Procedure Library (PL) files, and other such capabilities.

The ABL base plugin provides *Task Types* using which you can create tasks to perform various actions described in this guide. The name of task types gives an idea about what they are supposed to do. For more information, see [Task Types](#).

When writing tasks to build an ABL project, you may require to pull dependencies such as PL files from remote repositories. The base plugin provides configurations which can be used to manage dependencies. For more information, see [Dependency Management](#).

### Usage

To use the ABL base plugin, include the following in your build script (`build.gradle` file):

```
plugins {  
    id "progress.openedge.abl-base" version <version>  
}
```

### Task types

Create tasks using these task types:

#### ABLCompile

The ABLCompile task compiles ABL code.

#### Methods

Method	Description	Example
<code>exclude(String... excludes)</code>	Adds an ANT-style exclude pattern to this task.	<code>exclude('**/*.txt', '**/*.conf')</code>
<code>include(String... includes)</code>	Adds an ANT-style include pattern to this task.	<code>include('**/*.p', '**/*.cls')</code>
<code>source(String... sources)</code>	Adds a source to this task after the include and exclude patterns are applied. Sources can have an absolute or relative path to the root directory.	<code>source('src1', 'src2')</code>

Method	Description	Example
<code>propath(String... propaths)</code>	Sets the AVM's PROPATH. This method will be appended to the PROPATHs provided in ABLExtension	<code>propath('path1', "\${dlcHome}/path2")</code>
<code>avmOptions(Closure avmOptions)</code>	AVM options. The argument should be provided as a closure.  See <code>avmOptions</code> in the <a href="#">ABLCompile</a> section for more information.	<code>avmOptions{ tmpDir="path" tty.enabled=true }</code>
<code>dbConnection(Closure connection)</code>	Adds database configurations to this task.	<code>dbConnection{ dbName = 'sports2000' connectionParameters='-S 8000' }</code>
<code>dbConnectionReferenceId(String... refIds)</code>	Adds database task references.  See the <code>id</code> property of <a href="#">DBConnection</a> for more information.  You must add the database tasks' dependency to this task.	<code>dbConnectionReferenceId("id1", "id2")</code>
<code>compileOptions(Closure compileOptions)</code>	Compile options. The argument should be provided as a closure.  See the <code>compileOptions</code> section for more information.	<code>compileOptions{ multiCompile.enabled=true outputType="json" }</code>
<code>arguments(Map args)</code>	Additional arguments. These arguments are directly passed to PCT.	<code>arguments([k1:"v1", k2:'v2'])</code>

## Properties

Property	Required?	Description	Example	Default value
<code>openedgeVersion</code>	No (read-only property)	Prints the OpenEdge version. This can be used for logging. This is a read-only property.	None	None
<code>rcodeDir</code>	No	Sets the directory where the r-code will be generated.	<code>rcodeDir="path"</code>	<code>rcodeDir</code> set in ABLExtension.

Property	Required?	Description	Example	Default value
<code>propath</code>	No	Sets the PROPATH.	<code>propath = files('path1', "\${dlcHome}/path2")</code>	<code>propath</code> set in ABLExtension
<code>wrkDir</code>	No	Sets the working directory.	<code>wrkDir = 'path'</code>	<code>wrkDir</code> set in ABLExtension
<code>avmOptions{}</code>	No	AVM options. See the <code>avmOptions</code> section for more information.	None	<code>avmOptions</code> set in ABLExtension.
<code>compileOptions{}</code>	No	Compile options. See the <code>compileOptions</code> section for more information.	None	<code>compileOptions</code> set in ABLExtension.
<code>arguments</code>	No	Additional arguments. These arguments are directly passed to PCT.	<code>arguments = [k1: 'v1', k2: 'v2']</code>	None

**avmOptions{}**

Property	Description	Example	Default value
<code>tmpDir</code>	The temporary directory for the AVM run time. <code>-T</code> is the startup parameter option.	<code>tmpDir = 'path1'</code>	None
<code>tty.enabled</code>	The flag for using the <code>_progres</code> or <code>prowin</code> executables. Set to <code>true</code> for <code>_progres</code> , set to <code>false</code> for <code>prowin</code> (or <code>prowin32</code> for a 32-bit AVM).	<code>tty { enabled = 'true' }</code>	<code>true</code>
<code>xcodeSessionKey</code>	The XCODE session key for the security policy.	<code>xcodeSessionKey= 'myKey'</code>	None
<code>parameterFile</code>	The parameter file. <code>-pf</code> is the startup parameter option.	<code>parameterFile= 'pathToFile'</code>	None

Property	Description	Example	Default value
startupParameters	The startup parameters as a string. This will be ignored if a parameterFile is provided	startupParameters='-tok 4000 -s 200'	None
assembliesDir	The assemblies directory. -assemblies is the startup parameter option.	assembliesDir='path1'	None

### compileOptions{}

Property	Description	Example	Default value
xrefDir	The path where xref will be generated.	xrefDir='xref2'	'\${buildDir}xref'
debugListDir	The path where the debug listing will be generated.	debugListDir='debugList2'	'\${buildDir}/debugList'
preprocessDir	The path where the preprocess output will be generated.	preprocessDir='preprocess2'	'\${buildDir}/preprocess'
multiCompile.enabled	The COMPILER:MULTI-COMPILE attribute	multiCompile.enabled=true	false
strictOptions.requireFullNames	Specifies that the options require full names in the COMPILER statement. Value can be ignore or error.	strictOptions.requireFullNames='error'	ignore
strictOptions.requireFieldQualifiers	Specifies that the options require field qualifiers in the COMPILER statement. Value can be ignore or error.	strictOptions.requireFieldQualifiers='error'	ignore
strictOptions.requireFullKeywords	Specifies that the options require full keywords in the COMPILER statement. Value can be ignore or error.	strictOptions.requireFullKeywords='error'	ignore
strictOptions.requireReturnValues	Specifies that the options require return values in the COMPILER statement. Value can be ignore or error.	strictOptions.requireReturnValues='error'	ignore

Property	Description	Example	Default value
<code>listing.enabled</code>	The <code>LISTING</code> option in the <code>COMPILE</code> statement	<code>listing.enabled=true</code>	false
<code>xcodeKey</code>	The <code>XCODE</code> option in the <code>COMPILE</code> statement. Specify the expression as a string.	<code>xcodeKey= 'myKey '</code>	None
<code>xref.enabled</code>	The <code>XREF</code> option in <code>COMPILE</code> statement	<code>xref.enabled=true</code>	false
<code>xrefXml.enabled</code>	The <code>XREF-XML</code> option in the <code>COMPILE</code> statement. To enable this option, set the value to <code>true</code> .	<code>xrefXml.enabled=true</code>	false
<code>xrefString.enabled</code>	The <code>STRING-XREF</code> option in the <code>COMPILE</code> statement	<code>xrefString.enabled=true</code>	false
<code>xrefString.append</code>	The <code>APPEND</code> option in <code>STRING-XREF</code>	<code>xrefString.append=true</code>	false
<code>streamIO.enabled</code>	The <code>STREAM-IO</code> option in the <code>COMPILE</code> statement. To enable this option, set the value to <code>true</code> .	<code>streamIO.enabled=true</code>	false
<code>languages.list</code>	This is a comma-separated list of language segments to include in the compiled r-code.	<code>languages.list='lang1,lang2'</code>	None
<code>languages.textSegGrow</code>	The <code>TEXT-SEG-GROW</code> option. Set the growth-factor as an integer. Supported only when a language list is provided.	<code>languages.textSegGrow=10</code>	None
<code>debugList.enabled</code>	The <code>DEBUG-LIST</code> option in the <code>COMPILE</code> statement	<code>debugList.enabled=true</code>	false
<code>preprocess.enabled</code>	The <code>PREPROCESS</code> option in the <code>COMPILE</code> statement	<code>preprocess.enabled=true</code>	false
<code>v6Frame.enabled</code>	The <code>V6FRAME</code> option in the <code>COMPILE</code> statement	<code>v6Frame.enabled=true</code>	false
<code>v6Frame.useRevVideo</code>	The <code>USE-REVVIDEO</code> option in <code>V6FRAME</code>	<code>v6Frame.useRevVideo=true</code>	false



Property	Description	Example	Default value
<code>v6Frame.useUnderline</code>	The USE-UNDERLINE option in V6FRAME	<code>v6Frame.useUnderline=true</code>	false
<code>minSize.enabled</code>	The MIN-SIZE option in the COMPILER statement. To enable this option, set the value to true.	<code>minSize.enabled=true</code>	false
<code>outputType</code>	Use this option to change the output format. Available options are: json	<code>outputType= ' json '</code>	None

## Sample code snippet

The following code snippet is an example using ABLCompile:

```
abl{
    propath("${dlcHome}/tty/")
    avmOptions {}
    compileOptions{}
    rcodeDir='rcode1'
    wrkDir = "myWrkDir"

    dbConnection{
        dbName = 'sports2000'
        connectionParameters='-S 8000'
    }
}

task myDbConTask(type: DBConnection){
    id='idl'
    dbName='sports2020'
    parameterFile='pathToFile'
    aliases = ['alias1', 'alias2', 'aalias3']
}

task myCompileTask(type: ABLCompile){
    println "OE version: ${openedgeVersion}"

    source('src1', 'src2')
    include('**/*.p', '**/*.cls')
    exclude('**/*.txt', '**/*.conf')
    rcodeDir='rcode1'
    propath('path1', "${dlcHome}/path2")
    wrkDir = 'path1'
    avmOptions{
        tty.enabled='true'
        startupParameters='-tok 4000 -s 200'
    }
    compileOptions{
        xrefDir='xref2'
        preprocessDir=""
        preprocess.enabled=""
        strictOptions{
            requireFullNames='warning'
        }
        outputType='json'
    }

    dbConnection{
        dbName = 'sports2020'
        parameterFile='pathToParameterFile'
        connectionParameters='-S 8000'
        port = '8000'
        host = 'localhost'
        username = 'admin'
        password = 'admin2'
        aliases = ['a1', 'a2']
    }
    dbConnectionReferenceId("id0")
    dbConnectionReferenceId("id1", "id2")
}

myCompileTask.dependsOn myDbConTask
```

## ABLRun

Use this task to run ABL procedures.

## Methods

Method	Description	Example
<code>propath(String... propath)</code>	Sets the AVM's PROPATH. Will be appended to the PROPATHs provided in ABLExtension.	<code>propath('path1', "\${dlcHome}/path2")</code>
<code>profiler(Closure options)</code>	Enables profiling. See the <code>profiler</code> section for options	<code>profiler { enabled = true outputDir = "\${buildDir}/profiler" }</code>
<code>procedure(String procedure)</code>	Provides the procedure file to be run.	<code>procedure('src/main.p')</code>
<code>environment(String name, String value)</code>	Adds environment variables to pass to the system command	<code>environment('k1', 'v1')</code>
<code>environment(Map environment)</code>	Adds environment variables to pass to the system command	<code>environment([k1:"v2", k2:"v2"])</code>
<code>dbConnectionReferenceId(String... refIds)</code>	Adds database task references. Reference the <code>id</code> property of the <code>DBConnection</code> tasks for more information.  Note that you must add the database task dependency to this (compile task).	<code>dbConnectionReferenceId("id1", "id2")</code>
<code>dbConnection(Closure connection)</code>	Adds database configurations. The argument should be provided as a closure with its object following the structure supported by the <code>DBConnection</code> task type.  Note that braces can be removed following the Java/Groovy syntax.	<code>dbConnection{ dbName = 'sports2000' connectionParameters='-S 8000' }</code>

Method	Description	Example
<code>avmOptions(Closure avmOptions)</code>	AVM options. The argument should be provided as a closure. See the <code>avmOptions</code> section for more information.	<pre>avmOptions{   tmpDir="path"   tty.enabled=true }</pre>
<code>arguments(Map args)</code>	Additional arguments for ABL procedures. (These arguments are directly passed to PCT).	<code>arguments([k1:"v1", k2:'v2'])</code>

## Properties

Property	Required?	Description	Example	Default value
<code>dlcHome</code>	Yes  Should be set as described in the <code>ABLEExtension</code> section.	OpenEdge DLC path	None	None
<code>openedgeVersion</code>	No (read-only property)	Specifies the OpenEdge version. This can be used for logging. This is a read-only property.	None	None
<code>procedure</code>	Yes	Path of the procedure file that will be run.  Can be provided using the method <code>procedure(String procedure)</code> .	<code>procedure = 'src/main.p'</code>	None
<code>propath</code>	No	Sets the AVM's PROPATH.  Will be appended to the PROPATHs provided in <code>ABLEExtension</code> .	<code>propath = files('path1', "\${dlcHome}/path2")</code>	<code>propath</code> set in <code>ABLEExtension</code>
<code>wrkDir</code>	No	Working directory from where the AVM is started	<code>wrkDir = 'path'</code>	<code>wrkDir</code> set in <code>ABLEExtension</code>

Property	Required?	Description	Example	Default value
avmOptions	No	AVM options.  Use the method <code>avmOptions(Closure avmOptions)</code> to set the properties.  The properties defined in this task take priority if the same property is defined in <code>ABLEExtension</code>	None	<code>avmOptions</code> set in <code>ABLEExtension</code>
dbConnection	No	DB connection details.  Use methods <code>dbConnection(Closure connection)</code> or <code>dbConnection(String refIds)</code> to add	None	<code>dbConnections</code> set in <code>ABLEExtension</code>
environment	No	Environment variables to pass to the system command	<code>environment = [k1:"v2", k2:"v2"]</code>	None
arguments	No	Additional arguments for ABL procedures. (These arguments are directly passed to PCT.)	<code>arguments = [k1:"v1", k2:'v2']</code>	None

## Profiler

Property	Description	Example	Default value
enabled	Enables the profiler	<code>enabled=true</code>	false
description	Description of the profiler session	<code>description="description"</code>	None
outputDir	Generates a profiler output in this directory, with a unique name. Will be ignored if <code>outputFile</code> is provided.	<code>outputDir = 'path'</code>	<code>"\${buildDir}/profiler"</code>
outputFile	Profiler output file name	<code>outputFile='path'</code>	None

Property	Description	Example	Default value
coverage	Enables code coverage	coverage=true	false
statistics	Enables statistics	statistics=true	false
debugListDir	Generates debug listing files in this directory	debugListDir="path"	None

**avmOptions{}**

Property	Description	Example	Default value
tmpDir	The temporary directory for the AVM run time. -T is the startup parameter option.	tmpDir = 'path1'	None
tty.enabled	The flag for using the _progres or prowin executables. Set to true for _progres, set to false for prowin (or prowin32 for a 32-bit AVM).	tty { enabled = 'true' }	true
xcodeSessionKey	The XCODE session key for the security policy.	xcodeSessionKey='myKey'	None
parameterFile	The parameter file. -pf is the startup parameter option.	parameterFile='pathToFile'	None
startupParameters	The startup parameters as a string. This will be ignored if a parameterFile is provided	startupParameters='-tok 4000 -s 200'	None
assembliesDir	The assemblies directory. -assemblies is the startup parameter option.	assembliesDir='path1'	None

## Sample code snippet

The following code snippet is an example using ABLExtension:

```
task runMain(type: ABLRun){
    procedure = "src/main.p"
    proppath('src')

    wrkDir = "${buildDir}/rcode"
    avmOptions{
        tty.enabled = false
    }

    dbConnection{
        dbName="db/sports2000/sports2000"
        connectionParameters = "-1"
    }

    arguments = [ failOnError : true]
    profiler {
        enabled = true
        coverage = true
        outputDir = "${buildDir}/profiler"
        description = "Coverage for main.p"
    }
}
```

## ABLUnit

Use this task to run ABLUnit tests. This task type has all the methods and properties that are in [ABLRun](#) (except `procedure`). Additional properties and methods are listed in the following table:

### Methods

Method	Description	Example
<code>exclude(String... excludes)</code>	Adds an ANT-style exclude pattern	<code>exclude('**/*.txt', '**/*.conf')</code>
<code>include(String... includes)</code>	Adds an ANT-style include pattern	<code>include('**/*.p', '**/*.cls')</code>
<code>source(String... sources)</code>	Adds sources to this task, after the include and exclude patterns are applied.  The task is skipped if no sources are available.	<code>source('src1', 'src2')</code>

## Properties

Property	Required?	Description	Example	Default value
outputDir	No	Directory where the result file is placed.  Don't use this property under Linux, because a bug prevents results.xml from being generated	outputDir="path"	"\${projectDir}"

## APL

Use the APL task to create protected r-code archive file with .apl as extension.

This archive is based on \*.jar files, though any type of files can be added within it, AVM will access only the r-code and image files. For more information, see "Manage archive libraries" in the *Manage ABL Applications* guide.

The APL task has the following default properties:

- archiveExtension is set to .apl
- entryCompression is set to ZipEntryCompression.STORED because the AVM only supports uncompresses archive entries.

Some manifest attributes are mandated and some attribute values are defaulted. For more information, see "Access information in a manifest file" in the *Manage ABL Applications* guide.

The table below describes how APL task handles some of the attributes.

Attribute	Required	Default
'Implementation-Title'	No	archiveFileName
'Component-Name'	No	archiveFileName
'Package-Type'		apl (always)
'OpenEdge-Tool'		OpenEdge DevOps Framework (always)
'OpenEdge-Version'	No	abl.openedgeVersion.



## Sample code snippet 1

```
task createMyFirstProtectedRCodeArchive(type: APL) {
    from("${buildDir}/rcode")
    from("${buildDir}/resources")
    archiveBaseName = "myFirstProtectedRCode"
    destinationDirectory = project.file "$buildDir/dist"

    manifest {
        // only mandatory attribute, others are defaulted
        attributes.put "Implementation-Vendor", "Progress Software"
    }
}
```

## Sample code snippet 2

```
task createMyFirstProtectedRCodeArchive(type: APL) {
    from("${buildDir}/rcode")
    from("${buildDir}/resources")
    archiveBaseName = "myFirstProtectedRCode"
    destinationDirectory = project.file "$buildDir/dist"

    manifest {
        attributes.put 'Implementation-Title', 'My First ABL Application'
        attributes.put 'Implementation-Vendor', 'Progress Software'
        attributes.put 'Implementation-Version', '1.0.0'
        attributes.put 'Implementation-Vendor-ID', 'PRGS'
        attributes.put 'Component-Name', 'Core ABL'
        attributes.put 'Build-OS', 'unix'
        attributes.put 'Signature-Policy', 'required'
        attributes.put 'Validation-Policy', 'warn'

        attributes.put 'My-Custom-Attribute', 'DevOps Build'
    }
}
```

## Sample code snippet 3

```
task createMyFirstProtectedRCodeArchive(type: APL) {
    from("${buildDir}/rcode")
    from("${buildDir}/resources")
    archiveBaseName = "myFirstProtectedRCode"
    destinationDirectory = project.file "$buildDir/dist"

    manifest {
        // using APL's exposed Keys and Values
        attributes.put AttributeKeys.ImplementationTitle, 'My First ABL Application'
        attributes.put AttributeKeys.ImplementationVendor, 'Progress Software'
        attributes.put AttributeKeys.ImplementationVersion, '1.0.0'
        attributes.put AttributeKeys.ImplementationVendorID, 'PRGS'
        attributes.put AttributeKeys.ComponentName, 'Core ABL'
        attributes.put AttributeKeys.BuildOS, BuildOSValues.UNIX //
        BuildOSValues.WINDOWS / BuildOSValues.ALL
        attributes.put AttributeKeys.SignaturePolicy, SignaturePolicy.REQUIRED //
        SignaturePolicy.OPEN
        attributes.put AttributeKeys.ValidationPolicy, ValidationPolicy.WARN //
        ValidationPolicy.NONE / ValidationPolicy.FAIL

        attributes.put 'My-Custom-Attribute', 'My Awesome Build'
    }
}
```

## Extracting APL file

You can extract an APL file in a Gradle native way using `Copy` task. For example, refer to the following code snippet.

```
task extractMyFirstProtectedRCodeArchive(type: Copy) {  
    from zipTree("$buildDir/dist/myFirstProtectedRCode.apl")  
    into "$buildDir/extract/myFirstProtectedRCode"  
}
```

---

**Note:** You can update the content of an existing APL file by using the same APL task as used in the example above. For handling conflicts of duplicates, use the Gradle's JAR task type `duplicateStrategy` property.

---

Refer to the *Gradle* documentation for more information about `Copy` and `Jar` task types.

## BackupDB

Use this task to create a backup of an OpenEdge database.

### Properties

Property	Required?	Description	Example	Default value
source	Yes	Path to the database where the .db file is located.  Can be relative or absolute. May or may not include the .db file extension.	source="path"	None
target	Yes	Path of the backup file for the database.  Can be absolute or relative.	target="path"	None
overwrite	No	Whether to replace if the backup file exists	overwrite=true	false

Property	Required?	Description	Example	Default value
online	No	Indicates the database must be online or not when doing a backup.  If not set, the proutil -C holder is used to detect if the database is up.  Leave this out if you want the task to auto detect if this option should be used.	online=true	Auto
incremental	No	Performs an incremental backup.	incremental=true	false

### Sample code snippet

The following code snippet is an example using BackupDB:

```
task createDBBackup(type: BackupDB){
    source = "db/sports2020/sports2020.db"
    target = "${buildDir}/dist/sports2020.bck"
    overwrite = true
}
```

### CreateDB

Use this task to create an OpenEdge database.

### Methods

Method	Description	Example
arguments(Map args)	Additional arguments. These arguments are directly passed to PCT.	arguments( [k1: "v1", k2: 'v2' ] )

### Properties

Property	Required?	Description	Example	Default value
dbName	Yes	Database name	dbName= "dbName"	None
outputDir	No	The directory where the database will be created	outputDir="path"	"\${projectDir}"

Property	Required?	Description	Example	Default value
sourceDb	No	Copy the specified database to the target database.	sourceDb="path"	If attribute is not provided, the empty DB is used
schemaFile	No	Initial dump files to load into database. Separate dump filenames with commas. Files are resolved first as an absolute path, then relative to base directory. Wildcards are not expanded.	schemaFile="path"	None
structFile	No	Structure description file	structFile="path"	None
blockSize	No	Block size in kilobytes (1, 2, 4, or 8). Cannot be used with sourceDb attribute.	blockSize=2	8
tmpDir	No	The -T parameter when loading a schema	tmpDir="path"	None
cpInternal	No	The -cpinternal parameter when loading a schema	cpInternal="value"	None
newInstance	No	Appends -newInstance in the PROCOPY command line.	newInstance=true	false
largeFiles	No	Enable large files for this database.	largeFiles=true	false
arguments	No	Additional arguments for databases. (These arguments are directly passed to PCT.)	arguments = [k1:"v1", k2:'v2']	None

## Sample code snippet

The following code snippet is an example using CreateDB:

```
task createSports2000(type: CreateDB){
    dbName = 'sports2000'
    sourceDb = "${dlcHome}/sports2000"
    outputDir = "db/sports2000"
}
```

## DBConnection

The DBConnection task defines the OpenEdge database connection, which can be referred to by a compile task.

## Properties

Property	Required?	Description	Example	Default value
dbName	Yes, if parameterFile is not set	The physical database name (-db parameter). This can also be a full path.	dbName = 'sports2020'	None
parameterFile	Yes, if dbName is not set	The parameter file	parameterFile='pathToFile'	None
connectionParameters	Yes, if dbName and parameterFile are not set	The connection parameters as string. A temporary pf file is created and used as the parameterFile property.  This property will be ignored if <i>connection-name.parameterFile</i> is provided	connectionParameters='-S 8000'	None
port	No	The database port (-S parameter)	port = '8000'	None
host	No	The database host (-H parameter)	host = 'localhost'	None
username	No	The database user name (-U parameter)	username = 'admin'	None
password	No	The database user password (-P parameter)	password = 'admin'	None
aliases	No	The database aliases	aliases = ['alias1', 'alias2', 'aalias3']	None
id	Yes, if using as a task	The identifier for this task, which can be referred to from a compile task.	id = 'id1'	None

## Sample code snippet

The following code snippet is an example using DBConnection:

```
task conl(type: DBConnection){
    id = "idConl"
    dbName = 'sports2020'
    parameterFile='pathToFile'
    aliases = ['a1', 'a2']
}
```

### Note:

These properties are passed in the following order:

```
-db <dbName> -pf parameterFile> <other - attributes like -S 8655>
```

The properties that are defined later take priority, if the same parameter is repeated inside the `parameterFile`:

- If `-db` is defined inside the `pf` file and `dbName` is also set, then the property defined in the `pf` file takes priority.
- If any other parameter, such as `-S 8655`, is defined in the `pf` file and the corresponding attribute, such as `port='8665'`, is also set, then the attribute, `port`, takes priority.

Progress recommends that you do not define multiple database connection details, because this can lead to unexpected behavior.

## ExtractPL

Use this custom task to extract a PL file.

### Method

Method	Description	Example
<code>*from(String plFilePath</code>	Specify the <code>pl</code> path.	<code>from("./corelib.pl")</code>
<code>into(String destinationDir)</code>	Specify the destination directory for the extracted files.	<code>into("\$buildDir/corelib")</code>
<code>exclude(String... excludes)</code>	Adds an ANT style exclude pattern.	<code>exclude('**/*.p', '**/*.cls')</code>
<code>include(String... includes)</code>	Adds an ANT style include pattern.	<code>include('**/*.r', '**/*.conf')</code>
<code>*arguments(Map args)</code>	Additional arguments. <ul style="list-style-type: none"> <li>• <a href="https://ant.apache.org/manual/Types/fileset.html">https://ant.apache.org/manual/Types/fileset.html</a></li> <li>• <a href="https://ant.apache.org/manual/Types/Attribute.html">https://ant.apache.org/manual/Types/Attribute.html</a></li> </ul> These arguments are directly passed to PCT.	<code>arguments([dirmode:"0777", k2:'v2'])</code>

## Properties

Property	Required	Description	Example	Default Value
source	No	The file path to the PL which will be extracted.	source="./corelib.pl"	None
destinationDir	No	The destination directory for the extracted files.	destinationDir=\$buildDir/corelib	None
*overrideDuplicateStrategy	No	-	-	None
arguments	No	Additional arguments.  These arguments are directly passed to PCT.	arguments = [k1:"v1", k2:'v2']	None

## Sample Code Snippet

```
task extractCorelibPl(type: ExtractPL){
    from("./corelib.pl")
    into("$buildDir/corelib")
    include('**/*.r')
    arguments = [ dirmode:"0777", encoding : 'undefined' ]
}
```

## LoadDBSchema

Use this task to load an OpenEdge database schema file to a database.

## Methods

Method	Description	Example
exclude(String... excludes)	Adds an ANT-style exclude pattern.	exclude('**/*.txt', '**/*.conf')
include(String... includes)	Adds an ANT-style include pattern.	include('**/*.df')
source(String... sources)	The schema (.df) files to be loaded. Adds sources to this task after the include and exclude patterns are applied.  The task is skipped if no sources are present.	source('myDb.df', 'src/schema')

Method	Description	Example
<code>dbConnection(Closure connection)</code>	Adds database configurations. The argument should be provided as a closure with its object following the structure supported by DBConnection task type.  Note that braces can be removed following Java/Groovy syntax.	<code>dbConnection{ dbName = 'sports2000' connectionParameters='-S 8000' }</code>
<code>dbConnectionReferenceId(String... refIds)</code>	Adds database tasks' references. Reference the <code>id</code> property of the DBConnection tasks for more information.  Note that you must add the Database task dependency to this (compile task).	<code>dbConnectionReferenceId("id1", "id2")</code>
<code>arguments(Map args)</code>	Additional arguments for databases. (These arguments are directly passed to PCT.)	<code>arguments([k1: "v1", k2: 'v2'])</code>

## Properties

Property	Required?	Description	Example	Default value
<code>onlineChanges</code>	No	Relaxes requirements in order to apply online changes (use <code>SESSION:SCHEMA-CHANGE = 'NEW OBJECTS'</code> ).	<code>onlineChanges=true</code>	false

## Sample code snippet

The following code snippet is an example using LoadDBSchema:

```
task loadSports2000Schema(type: LoadDBSchema){
    source("myDb.df")
    onlineChanges = true
    dbConnection{
        dbName="db/sports2020/sports2020"
        connectionParameters = "-1"
    }
}
```

## PL

Use this task to create a Procedure Library (PL) file.



## Methods

Method	Description	Example
<code>from(String... from)</code>	Adds sources and r-codes to this task, after the include and exclude patterns are applied.	<code>from("\${buildDir}/rcode", 'src2')</code>
<code>exclude(String... excludes)</code>	Adds an ANT-style exclude pattern	<code>exclude('**/*.txt', '**/*.conf')</code>
<code>include(String... includes)</code>	Adds an ANT-style include pattern	<code>include('**/*.r', '**/*.p')</code>
<code>arguments(Map args)</code>	Additional arguments for libraries. (These arguments are directly passed to PCT.)	<code>arguments([k1:"v1", k2:'v2'])</code>

## Properties

Property	Required?	Description	Example	Default value
<code>plFile</code>	Yes, if <code>sharedFile</code> is not provided	The file path where the PL file is created	<code>plFile="path"</code>	None
<code>sharedFile</code>	Yes, if <code>plFile</code> is not provided	Memory mapped library to create the PL file	<code>sharedFile="path"</code>	None
<code>cpInternal</code>	No	Internal code page (-cpinternal parameter)	<code>cpInternal="value"</code>	None
<code>codepage</code>	No	Name of the code page for the library	<code>codepage="value"</code>	None
<code>arguments</code>	No	Additional arguments for libraries. (These arguments are directly passed to PCT.)	<code>arguments = [k1:"v1", k2:'v2']</code>	None

## Sample code snippet

The following code snippet is an example using PL:

```
task createPL2(type: PL){
    plFile = "${buildDir}/dist/test.pl"
    from("${buildDir}/rcode")
    include('**/*.r')
    arguments = [encoding : 'undefined' ]
}
```

## Sample custom task

The following sample code demonstrates how you can use the ABLCompile and DBConnection task types in a custom task:

```

import com.progress.gradle.abl.tasks.*
plugins {
    id "progress.openedge.abl-base" version "2.2"
}

// This section defines the global configuration available to all task types.

abl{
    proppath("${dlcHome}/tty/")
    avmOptions {}
    compileOptions{}
    rcodeDir='rcode1'
    wrkDir = "myWrkDir"

    dbConnection{
        dbName = 'sports2000'
        connectionParameters='-S 8000'
    }
}

// This section defines the DBConnection task.

task myDbConTask(type: DBConnection){
    id='idl'
    dbName='sports2020'
    parameterFile='pathToFile'
    aliases = ['alias1', 'alias2', 'aalias3']
}

// This section defines the ABLCompile task.

task myCompileTask(type: ABLCompile){
    println "OE version: ${openedgeVersion}"

    source('src1', 'src2')
    include('**/*.p', '**/*.cls')
    exclude('**/*.txt', '**/*.conf')
    rcodeDir='rcode1'
    proppath('path1', "${dlcHome}/path2")
    wrkDir = 'path1'
    avmOptions{
        tty.enabled='true'
        startupParameters='-tok 4000 -s 200'
    }
    compileOptions{
        xrefDir='xref2'
        preprocessDir=""
        preprocess.enabled=""
        strictOptions{
            requireFullNames='warning'
        }
        outputType='json'
    }
}

dbConnection{
    dbName = 'sports2020'
    parameterFile='pathToParameterFile'
    connectionParameters='-S 8000'
    port = '8000'
    host = 'localhost'
    username = 'admin'
    password = 'admin2'
    aliases = ['alias1', 'alias2', 'aalias3']
}
dbConnectionReferenceId("id0")
dbConnectionReferenceId("idl", "id2")
}

myCompileTask.dependsOn myDbConTask

```

You can find more sample projects on the [Progress community](#).

## Task types for Progress Application Server (PAS)

You can generate packages that can be deployed to PAS for OpenEdge using the following task types:

### Oear

The ABL `App` level package task is used to generate an OpenEdge Application Archive (`Oear`) file, which is a specific type of ZIP file that contains web application resources that can be deployed to a PAS instance in a single operation.

`Oear` extends the `Zip` task type of Gradle and supports all the properties and methods supported by `Zip` task type. The following tables describe the extra configurations added by `Oear` task type along with some common configurations inherited from the `Zip` task type of Gradle.

### Methods

Method	Description	Example
<code>tlr(configureClosure)</code>	Adds content to the <code>tlr/</code> archive directory. Execute the given closure configuration to configure <code>org.gradle.api.file.CopySpec</code>	<pre>tlr {   from 'src/main/tlr'   include     '**/*.properties'   exclude '**/*.txt' }</pre>
<code>webapps(configureClosure)</code>	Adds content to the <code>webApps/</code> archive directory. Execute the given closure configuration to configure <code>org.gradle.api.file.CopySpec</code>	<pre>webapps { from 'src/main/webapps' }</pre>
<code>openedge(configureClosure)</code>	Adds content to the <code>openedge/</code> archive directory. Execute the given closure configuration to configure <code>org.gradle.api.file.CopySpec</code>	<pre>openedge {   from 'src/main/openedge'   include '**/*.r'   exclude '**/*.txt' }</pre>
<code>conf(configureClosure)</code>	Adds content to the <code>conf/</code> archive directory. Execute the given closure configuration to configure <code>org.gradle.api.file.CopySpec</code>	<pre>conf { from 'src/main/conf' }</pre>

Method	Description	Example
<code>bin(configureClosure)</code>	<b>Optional.</b> Adds content to the bin/archive directory.  Execute the given closure configuration to configure <code>org.gradle.api.file.CopySpec</code>	<pre>bin { from 'src/main/bin' }</pre>
<code>instance(configureClosure)</code>	<b>Optional.</b> Adds content to the instance/archive directory.  Execute the given closure configuration to configure <code>org.gradle.api.file.CopySpec</code>	<pre>instance { from 'src/main/instance' }</pre>
<code>manifest(configureClosure)</code>	Configures the manifest for Oear archive type.  Execute the given closure configuration to configure the manifest property.	<pre>manifest {   attributes   "Implementation-Title": "My ABL Application"   attributes   "Implementation-Version": "1.0.0"   from   ("&lt;another-manifest-file-path&gt;") }</pre>

## Properties

Property	Required?	Description	Example	Default value
<code>ablAppName</code>	Yes	The name of the ABL application.	<code>ablAppName = "myABLApp"</code>	None
<code>archiveBaseName</code>	No	The base name of the archive.	<code>archiveBaseName = "myABLApp"</code>	<code>ablAppName</code>
<code>archiveAppendix</code>	No	The appendix part of the archive name, if any.	<code>archiveAppendix = "appendixName"</code>	None
<code>archiveVersion</code>	No	The version part of the archive name.	<code>archiveVersion = "1.0.0"</code>	None
<code>archiveClassifier</code>	No	The classifier part of the archive name, if any.	<code>archiveClassifier = "source"</code>	None

Property	Required?	Description	Example	Default value
archiveExtension	No	The extension type used for the archive.	archiveExtension = "customExt"	oear
archiveFileName	No	Displays the archive name in the following format:  [archiveBaseName] -[archiveAppendix] -[archiveVersion] -[archiveClassifier] .[archiveExtension]	archiveFileName = "myArchive.customExt"	\${ablAppName}.oear
destinationDirectory	No	The directory where the archive is placed.	destinationDirectory = project.file("\${buildDir}/dist")	project.distsDir

Property	Required?	Description	Example	Default value
manifest	No	<p>The manifest for the Oear archive, which is placed in the conf/ folder.</p> <p>You can configure it the same way as you configure</p> <pre>org.gradle.api.java.archives.Manifest</pre> <hr/> <p><b>Note:</b> The format for Build-Date is <code>DateTimeFormatter.ISO_OFFSET_DATE_TIME</code>.</p> <hr/>	<pre>manifest.attributes ImplementationTitle:   "My Server ABL   Application" manifest.attributes ImplementationVersion:   "1.0.0"</pre>	<pre>Manifest-Version:   1.0 ABL-Application-Name:   \${ablAppName} ABL-Application-Version:   0.0.0.0 Package-Type:   oear Build-Date:   &lt;build-date-time&gt; OpenEdge-Tool:   OpenEdge   DevOps   Framework</pre>
instanceDirectory	No	<p>Exports the ABL application from an existing PAS for OpenEdge instance location.</p> <p>An ABL App with the name <code>ablAppName</code> should exist for the instance.</p> <p>When you use the <code>instanceDirectory</code> property in combination with folder copies, the final archive appends the contents. For more information, refer to <i>Sample code snippet 2</i>.</p>	<pre>instanceDirectory.project.file "\${WRK}/myABLApp"</pre>	None
fullExport	No		<pre>fullExport = false</pre>	true

Property	Required?	Description	Example	Default value
		<p>Determines whether to perform a full export or incremental export of web applications present as part of the ABL App.</p> <hr/> <p><b>Note:</b> The <code>fullExport</code> property is applicable only when <code>instanceDirectory</code> is specified.</p> <hr/>		

### Sample code snippet 1

The following code snippet is an example of using the `Oear` task type :

```
task createABLAppArchive(type: Oear){
    ablAppName = "myABLApp"
    destinationDirectory = project.file "${buildDir}/dist" //will create 'myABLApp.oear'
    file at this location
    tlr {
        from 'src/main/tlr'
        include '**/*.properties'
        exclude '**/*.txt'
    }
    webapps { from 'src/main/webapps' }
    openedge {
        from 'src/main/openedge'
        include '**/*.r'
        exclude '**/*.txt'
    }
    conf {
        from 'src/main/conf'
        exclude '**/*.MF' //exclude direct copy of manifest file and append using manifest
        section
    }
    manifest {
        attributes "Implementation-Title": "My ABL Application"
        attributes "Implementation-Version": "1.0.0"
        from ("src/main/conf/MANIFEST.MF")
    }
}
```

### Sample code snippet 2

The following code snippet is an example of using the `Oear` task type with the `instanceDirectory` property configured.

```
task createABLAppArchiveFromInstance(type: Oear){
    ablAppName = "oepas1"
    instanceDirectory = project.file "C:/OpenEdge/WRK/oepas1"
    fullExport = false
    destinationDirectory = project.file "${buildDir}/dist" //will create 'oepas1.oear'
    file at this location
}
```



```
manifest {
    attributes "Implementation-Title": "My ABL Application"
    attributes "Implementation-Version": "1.0.0"
}
```

## OEWar

The OEWar task is used to package an ABL web application project and generate a WAR file.

OEWar task type extends from the War task type of Gradle, and supports all the properties and methods supported by the War task type. The following tables describe the extra configurations added by OEWar along with some common configurations inherited from the War task type of Gradle.

### Methods

Method	Description	Example
<code>services(serviceNames)</code>	Specifies the service names that should be exported. By default, all the services are exported.  <b>Note:</b> This method is applicable only when <code>projectLocation</code> property is provided.	<pre>services("service1") services("service2", "service3")</pre>
<code>webInf(configureClosure)</code>	Adds the content to the WEB-INF/archive directory.  Execute the given closure configuration to configure the <code>org.gradle.api.file.CopySpec</code>	<pre>webInf { from 'src/main/webInf' } webInf {     from("src/openedge")     into("openedge")     include "**/*.r"     exclude "**/*.txt" }</pre>
<code>manifest(configureClosure)</code>	Configures the manifest for OEWar archive type.  Execute the given closure configuration to configure the manifest property.	<pre>manifest {     attributes     "Implementation-Title": "My ABL Web Application"     attributes     "Implementation-Version":     "1.0.0"     from     ("&lt;another-manifest-file-path&gt;") }</pre>

### Properties

Property	Required?	Description	Example	Default value
<code>webAppName</code>	Yes	The name of the ABL web application.	<code>webAppName = "myABLWebApp"</code>	None

Property	Required?	Description	Example	Default value
<code>archiveBaseName</code>	No	The base name of the archive.	<code>archiveBaseName = "myABLWebApp"</code>	<code>webAppName</code>
<code>archiveAppendix</code>	No	The appendix part of the archive name, if any.	<code>archiveAppendix = "appendixName"</code>	None
<code>archiveVersion</code>	No	The version part of the archive name.	<code>archiveVersion = "1.0.0"</code>	None
<code>archiveClassifier</code>	No	The classifier part of the archive name, if any.	<code>archiveClassifier = "source"</code>	None
<code>archiveExtension</code>	No	The extension used for the <code>OEWar</code> archive type.	<code>archiveExtension = "customExt"</code>	<code>war</code>
<code>archiveFileName</code>	No	Displays the archive name in the following format:  [ <code>archiveBaseName</code> ] -[ <code>archiveAppendix</code> ] -[ <code>archiveVersion</code> ]-[ <code>archiveClassifier</code> ] .[ <code>archiveExtension</code> ]	<code>archiveFileName = "myABLWebApp-1.0.0-source-customExt"</code>	<code>\${webAppName}.war</code>
<code>destinationDirectory</code>	No	The directory where the archive is placed.	<code>destinationDirectory = project.file("\${buildDir}/dist")</code>	<code>project.distsDir</code>

Property	Required?	Description	Example	Default value
manifest	No	<p>The manifest for the OEWar archive, which is placed in the WEB-INF/ folder.</p> <p>You can configure it the same way as you configure</p> <pre>org.gradle.api.java.archives.Manifest</pre> <hr/> <p><b>Note:</b> The format for Build-Date is</p> <pre>DateTimeFormatter.ISO_OFFSET_DATE_TIME</pre> <hr/>	<pre>manifest.attributes "Implementation-Title": "My Server ABL Application" manifest.attributes "Implementation-Version": "1.0.0"</pre>	<pre>Manifest-Version: 1.0 ABL-Web-Application-Name: \${webAppName} ABL-Web-Application-Version: 0.0.0.0 Package-Type: war Build-Date: &lt;build-date-time&gt; OpenEdge-Tool: OpenEdge DevOps Framework</pre>
projectLocation	No	<p>Specifies the path with all the relevant files and folder to the Progress Developer Studio for OpenEdge server project. For example,</p> <pre>.services/ , PASOEContent/, and so on.</pre>	<pre>projectLocation.project.file "&lt;path-to-project&gt;"</pre>	None
serviceList	No	<p>Specifies the service names that should be exported. By default, all the services are exported.</p> <hr/> <p><b>Note:</b> This property is applicable only when the projectLocation is specified.</p> <hr/>	<pre>servicesList = ["service1", "service2"]</pre>	None
verbose	No	<p>Use this property to enable logs.</p>	<pre>verbose = true</pre>	false

## Sample code snippet 1

The following code snippet is an example of using the `OEWAr` task type from a Progress Developer Studio for OpenEdge project:

```
task createWebAppArchive(type: OEWAr){
    webAppName = "myABLWebApp"
    projectLocation = project.file "./"
    verbose = true
    destinationDirectory = project.file "${buildDir}/dist"
    manifest {
        attributes "Implementation-Title": "My ABL Web Application"
        attributes "Implementation-Version": "1.0.0"
        from ("PASOEContent/META-INF/MANIFEST.MF")
    }
}
```

## Sample code snippet 2

The following code snippet is another example using the `OEWAr` task type from a Progress Developer Studio for OpenEdge project:

```
task createWebAppArchive(type: OEWAr){
    webAppName = "myABLWebApp"
    projectLocation = project.file "./"
    services("service2", "service3")
    verbose = true
    destinationDirectory = project.file "${buildDir}/dist"
    manifest {
        attributes "Implementation-Title": "My ABL Web Application"
        attributes "Implementation-Version": "1.0.0"
        from ("PASOEContent/META-INF/MANIFEST.MF")
    }
}
```

## Sample code snippet 3

The following code snippet is another example using the `OEWAr` task type from a custom project or folder structure:

```
task createWebAppArchive(type: OEWAr){
    webAppName = "myABLWebApp"
    destinationDirectory = project.file "${buildDir}/dist"

    //add the template provided in DLC
    from("${abl.dlcHome}/servlets/oeabl") {
        exclude "<files-that-should-be-replaced>"
    }

    //fail for duplicates
    duplicatesStrategy = DuplicatesStrategy.FAIL

    //add folders/files inside the archive
    from("src/static") {
        into("static")
        include "**/*.js"
        include "**/*.json"
    }
    webInf {
        from("src/openedge")
        into("openedge")
        include "**/*.r"
        exclude "**/*.txt"
    }
    webInf {
```

```

    from("src/tlr")
    into("tlr")
  }

  //configure manifest
  manifest {
    attributes "ABL-Web-Application-Name": "myABLWebApp"
    attributes "ABL-Web-Application-Version": "0.0.0.0"
    attributes "Implementation-Title": "My ABL Web Application"
    attributes "Implementation-Version": "1.0.0"
    from ("<path-to-manifest-file>")
  }
}

```

## OESvcZip

The `OESvcZip` task type is used to create incremental ZIP file for REST or WEB service package that can be deployed to a PAS for OpenEdge instance in a single operation.

`OESvcZip` extends from the `OEwar` task type and hence supports all the properties and methods supported by the `OEwar` task type. The following tables describe the extra configurations added by the `OESvcZip` task type.

### Methods

Method	Description	Example
<code>services(serviceNames)</code>	Specifies the service names that should be exported. By default, all the services are exported.	<pre> services("service1") services("service2", "service3") </pre>
<code>manifest(configureClosure)</code>	Configures the manifest for the <code>oesvczip</code> archive type.	<pre> manifest {   attributes     "Implementation-Title": "My   ABL Application"   attributes     "Implementation-Version":     "1.0.0"   from     ("&lt;another-manifest-file-path&gt;") } </pre>

### Properties

Property	Required?	Description	Example	Default value
<code>webAppName</code>	Yes	The name of the ABL web application.	<code>webAppName = "myABLWebApp"</code>	None
<code>archiveBaseName</code>	No	The base name of the archive.	<code>archiveBaseName = "myABLWebApp"</code>	<code>webAppName</code>

Property	Required?	Description	Example	Default value
<code>archiveAppendix</code>	No	The appendix part of the archive name, if any.	<code>archiveAppendix = "appendixName"</code>	None
<code>archiveVersion</code>	No	The version part of the archive name.	<code>archiveVersion="1.0.0"</code>	None
<code>archiveClassifier</code>	No	The classifier part of the archive name, if any.	<code>archiveClassifier="sure"</code>	None
<code>archiveExtension</code>	No	The extension used for the OESvcZip archive type.	<code>archiveExtension="custom"</code>	zip
<code>archiveFileName</code>	No	Displays the archive name in the following format:  [archiveBaseName] -[archiveAppendix] -[archiveVersion] -[archiveClassifier] .[archiveExtension]	<code>archiveFileName="archiveName- archiveBaseName- archiveVersion- archiveClassifier- archiveExtension"</code>	<code>\${webAppName}.zip</code>
<code>destinationDirectory</code>	No	The directory where the archive is placed.	<code>destinationDirectory=project.file "\${buildDir}/dist"</code>	<code>project.distsDir</code>

Property	Required?	Description	Example	Default value
manifest	No	<p>The manifest for the OESvcZip archive, which is placed in the WEB-INF/ folder.</p> <p>You can configure it the same way as you configure <code>org.gradle.api.plugins.archives.Manifest</code></p> <hr/> <p><b>Note:</b> The format for Build-Date is <code>DateTimeFormatter.ISO_OFFSET_DATE_TIME</code></p> <hr/>	<pre>manifest.attributes "Implementation-Title": "My Server ABL Application" manifest.attributes "Implementation-Version": "1.0.0"</pre>	<pre>Manifest-Version: 1.0 Package-Type: zip Build-Date: &lt;build-date-time&gt; OpenEdge-Tool: OpenEdge DevOps Framework</pre>
projectLocation	No	Specifies the path to the Progress Developer Studio for OpenEdge server project with all relevant files and folders. For example, <code>.service/</code> , <code>PASOEContent/</code> , and so on.	<pre>projectLocation = project.file "&lt;path-to-project&gt;"</pre>	None
serviceList	No	Specifies the service names that should be exported. By default, all the services are exported.	<pre>servicesList = ["service1", "service2"]</pre>	None
verbose	No	Use this property to enable logs.	<code>verbose = true</code>	false

### Sample code snippet 1

The following code snippet is an example of using the OESvcZip task type:

```
task createSvcZipArchive(type: OESvcZip) {
    webAppName = "myABLWebApp"
    projectLocation = project.file "./"
    verbose = true
    destinationDirectory = project.file "${buildDir}/dist"
    manifest {
        attributes "Implementation-Title": "My Incremental Archive"
        attributes "Implementation-Version": "1.0.0"
    }
}
```

```

    }
    from ( "PASOEContent/META-INF/MANIFEST.MF" )
}

```

## Sample code snippet 2

The following code snippet is another example of using the `OESvcZip` task type:

```

task createSvcZipArchive(type: OESvcZip){
    webAppName = "myABLWebApp"
    projectLocation = project.file "./"
    services("service2", "service3")
    verbose = true
    destinationDirectory = project.file "${buildDir}/dist"
    manifest {
        attributes "Implementation-Title": "My Incremental Archive"
        attributes "Implementation-Version": "1.0.0"
        from ( "PASOEContent/META-INF/MANIFEST.MF" )
    }
}

```

## Oeds

An ABL service is a set of business logic that you can access using an URL path. The `Oeds` task type allows you to package the ABL services into one deployment action. This package allows you to build a custom service quickly and to easily deploy to a PAS for OpenEdge instance. This package creates a single `Oeds` archive file for each transport, which makes the ABL Service easy to maintain and benefit from CI/CD processes

The `Oeds` task type extends from the `Zip` task type of Gradle and supports all the properties and methods supported by it. The following tables describe the extra configurations added by the `Oeds` task type along with some common configurations inherited from the `Zip` task type of Gradle.

## Methods

Method	Description	Example
<code>conf(configurationClosure)</code>	<p>Adds the content to the <code>conf/</code> directory of the <code>oeds</code> archive.</p> <p>Execute the given closure configuration to configure the <code>org.gradle.api.file.CopySpec</code></p>	<pre>conf { from 'src/main/conf' }</pre>



Method	Description	Example
<code>static(configureCopySpec)</code>	<p>Adds the contents to the <code>static/directory</code> of the <code>oeds</code> archive.</p> <p>Execute the given closure configuration to <code>configureorg.gradle.api.file.CopySpec</code></p> <p>Adds the content of various files such as, <code>**/*.js</code>, <code>**/*.json</code>, <code>**/*.xml</code>, <code>**/*.css</code>, <code>**/*.png</code>, and <code>**/*.*</code> to the <code>static/</code> archive directory.</p> <hr/> <p><b>Note:</b> <code>static</code> is a special Java keyword. Pay attention to the <code>_</code> appended as suffix to the <code>static</code> method.</p> <hr/>	<pre>static_{ from 'src/main/static' }</pre>
<code>openedge(configureCopySpec)</code>	<p>Adds the content of various files such as <code>**/*.r</code>, <code>**/*.cls</code>, <code>**/*.p</code>, <code>**/*.w</code>, and <code>**/**</code> to the <code>openedge/</code> directory of the <code>Oeds</code> archive.</p> <p>Execute the given closure configuration to <code>configureorg.gradle.api.file.CopySpec</code></p>	<pre>openedge {   from   'src/main/openedge'   include '**/*.r'   exclude '**/*.txt' }</pre>
<code>tlr(configureCopySpec)</code>	<p>Adds the content of <code>build.xml</code>, <code>merge.properties</code>, and <code>merge.oeablSecurity</code> to the <code>tlr/</code> directory of the <code>Oeds</code> archive.</p>	<pre>tlr {   from 'src/main/tlr'   include   '**/*.properties'   exclude '**/*.txt' }</pre>
<code>svc(configureCopySpec)</code>	<p>Adds the content of <code>\${oepas-ablsvc}.paar</code>, <code>\${oepas-ablsvc}.wsm</code>, <code>\${oepas-ablsvc}.map/gen</code>, <code>\${oepas-ablsvc}.handlers</code>, and <code>\${oepas-ablsvc}.json</code> to the <code>svc/</code> directory of the <code>Oeds</code> archive.</p>	<pre>svc { from 'src/main/svc' }</pre>

Method	Description	Example
<code>custom(configureCopySpec)</code>	Adds the content of *. * file to the custom/ directory of Oeds archive.	<pre>custom { from 'src/main/custom' }</pre>
<code>manifest(configureClosure)</code>	<p>Configures the manifest for Oeds archive type.</p> <p>Execute the given closure configuration to configure the manifest property.</p>	<pre>manifest {   attributes   "Implementation-Title": "My   ABL Service"   attributes   "Implementation-Version":   "1.0.0"   from   ("&lt;another-manifest-file-path&gt;") }</pre>

## Properties

Property	Required?	Description	Example	Default value
<code>serviceName</code>	Yes	The name of the ABL Service.	<code>serviceName="myABService"</code>	None
<code>serviceType</code>	Yes	<p>The type of ABL service used. It can be one of the following:</p> <ul style="list-style-type: none"> <li>• REST</li> <li>• SOAP</li> <li>• WEB</li> <li>• STATIC</li> <li>• APSV</li> </ul>	<code>serviceType="REST"</code>	None
<code>archiveBaseName</code>	No	The base name of the archive.	<code>archiveBaseName="myABService"</code>	<code>serviceName</code>
<code>archiveAppendix</code>	No	The appendix part of the archive name, if any.	<code>archiveAppendix="appendix"</code>	None
<code>archiveVersion</code>	No	The version part of the archive name.	<code>archiveVersion="1.0.0"</code>	None
<code>archiveClassifier</code>	No	The classifier part of the archive name, if any.	<code>archiveClassifier="source"</code>	None

Property	Required?	Description	Example	Default value
archiveExtension	No	The extension used for the Oeds archive type.	archiveExtension="custom"	oeds
archiveFileName	No	Displays the archive name in the following format:  [archiveBaseName] -[archiveAppendix] -[archiveVersion]-[archiveClassifier] .[archiveExtension]	archiveFileName="Archive-\${archiveBaseName}-\${archiveAppendix}-\${archiveVersion}-\${archiveClassifier}.\${archiveExtension}"	\${serviceName}.oeds

Property	Required?	Description	Example	Default value
destinationDirectory	No	The directory where the archive is placed.	destinationDirectory = project.file("\${buildDir}/dist")	project.distsDir
manifest	No	<p>The manifest for the Oeds archive, which is placed in the conf/ folder.</p> <p>You can configure it the same way as you configure <code>org.gradle.api.java.archives.Manifest</code></p> <hr/> <p><b>Note:</b> The format for Build-Date is <code>DateTimeFormatter.ISO_OFFSET_DATE_TIME</code></p> <hr/>	<pre>manifest.attributes {     "Implementation-Title": "My Server ABL Application" } manifest.attributes {     "Implementation-Version": "1.0.0"</pre>	<pre>Manifest-Version: 1.0 ABL-Service-Name: \${serviceName} ABL-Service-Type: \${serviceType} ABL-Service-Version: 0.0.0.0 Package-Type: oeds Build-Date: &lt;build-date-time&gt; OpenEdge-Tool: OpenEdge DevOps Framework</pre>

## Sample code snippet

The following code snippet is an example using the Oeds task type:

```
task createServiceArchive(type: Oeds){
    serviceName = "myABLSERVICE"
    serviceType = "REST"
    destinationDirectory = project.file "${buildDir}/dist"
    conf {
        from 'src/main/conf'
        exclude '**/*.MF' //exclude direct copy of manifest file and append using manifest section
    }
    static_ { from 'src/main/static' }
    openedge {
        from 'src/main/openedge'
        include '**/*.r'
        exclude '**/*.txt'
    }
    tlr {
        from 'src/main/tlr'
        include '**/*.properties'
        exclude '**/*.txt'
    }
    svc { from 'src/main/svc' }
    custom { from 'src/main/custom' }
    manifest {
        attributes "Implementation-Title": "My ABL Service"
        attributes "Implementation-Version": "1.0.0"
        from ("src/main/conf/MANIFEST.MF")
    }
}
```

## Paar

The Paar task type is used for packaging REST services.

The `Paar` task type expects the availability of PAS for OpenEdge server project with all relevant files and folders such as `.service/`, `PASOEContent/`, and so on.

`Paar` extends from `Jar` task type of Gradle and hence supports all the properties and methods supported by `Jar` task type. The following tables describe the extra configurations added by the `Paar` task type along with some common configurations inherited from the `Jar` task type of Gradle.

## Methods

Method	Description	Example
<code>manifest(configureClosure)</code>	Configures the manifest for the <code>Paar</code> archive type.  Execute the given closure configuration to configure the manifest property.	<pre>manifest {     attributes     "Implementation-Title": "My     ABL Service"     attributes     "Implementation-Version":     "1.0.0"     from     ("&lt;another-manifest-file-path&gt;") }</pre>

## Properties

Property	Required?	Description	Example	Default value
<code>serviceName</code>	Yes	The name of the ABL service.	<code>serviceName="ABLServic</code>	None
<code>archiveBaseName</code>	No	The base name of the archive.	<code>archiveBaseName="ABLServic</code>	<code>serviceName</code>
<code>archiveAppendix</code>	No	The appendix part of the archive name, if any.	<code>archiveAppendix="appendix"</code>	None
<code>archiveVersion</code>	No	The version part of the archive name.	<code>archiveVersion</code> <code>= "1.0.0"</code>	None
<code>archiveClassifier</code>	No	The classifier part of the archive name, if any.	<code>archiveClassifier</code> <code>= "source"</code>	None
<code>archiveExtension</code>	No	The extension used for the <code>Paar</code> archive type.	<code>archiveExtension</code> <code>= "customExt"</code>	<code>paar</code>

Property	Required?	Description	Example	Default value
archiveFileName	No	Displays the archive name in the following format:  [archiveBaseName] -[archiveAppendix] -[archiveVersion]-[archiveClassifier] .[archiveExtension]	<code>archiveFileName=Archive-\${serviceName}.aar</code>	<code>\${serviceName}.aar</code>
destinationDirectory	Yes	The directory where the archive is placed.	<code>destinationDirectory=project.buildDir</code> <code>"\${buildDir}/dist"</code>	<code>project.distsDir</code>
manifest	No	The manifest for the Paar archive, which is placed in the WEB-INF/ folder.  You can configure it the same way as you configure <code>org.gradle.api.plugins.archives.Manifest</code>  <b>Note:</b> The format for Build-Date is <code>DateTimeFormatter.ISO_OFFSET_DATE_TIME</code>	<code>manifest.attributes</code>  "Implementation-Title": "My Server ABL Application" <code>manifest.attributes</code>  "Implementation-Version": "1.0.0"	Manifest-Version: 1.0 ABL-Service-Name: <code>\${serviceName}</code> ABL-Service-Version: 0.0.0.0 ABL-Service-Type: REST Package-Type: paar Build-Date: <build-date-time> OpenEdge-Tool: OpenEdge DevOps Framework
projectLocation	Yes	Specifies the path to the Progress Developer Studio for OpenEdge server project.	<code>projectLocation=project.projectDir</code> <code>"&lt;path-to-project&gt;"</code>	None
verbose	No	Use this property to enable logs.	<code>verbose = true</code>	false

## Sample code snippet

The following code snippet is an example using the `paar` task type:

```
task createPaar(type: Paar) {
    serviceName = "myABLRestService"
    projectLocation = project.file "/"
    destinationDirectory = project.file "$buildDir/dist"
    verbose = true
    manifest {
        attributes "Implementation-Title": "My ABL REST Service"
        attributes "Implementation-Version": "1.0.0"
        from ("PASOEContent/META-INF/MANIFEST.MF")
    }
}
```

## Dependency management

The ABL base plugin provides configurations which can be used to manage `PL` file dependencies. Further sections provide the list of configurations that handle `PL` dependencies.

### pl

Use this configuration to specify `PL` files dependencies containing only `r-code`. The `PL` files automatically download from a repository. These dependencies can then be used to configure the `PROPATH` of the AVM in different tasks (for example, while creating compile tasks using `ABLCompile`).

#### Example

```
//declare repositories

repositories {

    maven {
        url = '<maven repo url>'
        metadataSources {
            artifact()//needed to download artifacts (like .pl file) without a pom file in the
            repository
        }
    }
}

//define the dependencies
dependencies {
    pl("progress.openedge.oedf.test.apps:avengers:1.0.0@pl")
}
task myCompileTask1(type: ABLCompile){
    source("src/")
    propath(files(configurations.pl.files))
}
```

The `configurations.pl.files` returns a set of `PL` files defined using the `pl` configuration and these `PL` files are directly added to the **PROPATH**. AVM cannot recognize non `r-code` files inside the `PL` and you should use `plSource` to manage such dependencies.

### pl source

Use the `plSource` configuration to specify `PL` files containing source files (such as ABL sources, image files, config files, and so on). The `PL` files are automatically downloaded from a repository and then extracted using the `ExtractPLTransform` transform. Use the extracted directory of these dependencies to configure the **PROPATH** of the AVM in different tasks (for example, while creating compile tasks using `ABLCompile`).

## Example

```
//declare repositories
repositories {
    maven {
        url = '<maven repo url>'
        metadataSources {
            artifact()//needed to download artifacts (like .pl file) without a pom file in the
            repository
        }
    }
}

//define the dependencies
dependencies {
    pl("progress.openedge.oedf.test.apps:avengers:1.0.0@pl")
    plSource("progress.openedge.oedf.test.apps:avengers:1.0.0:sources@pl")
}

task myCompileTask2(type: ABLCompile){
    source("src/")
    propath(files(configurations.pl.files))
    propath(files(configurations.plSource.files))
}
```

The `configurations.plSource.files` returns a set of directories where PL files defined using the `plSource` configuration are extracted. Thus, the paths to extracted directories are added to the **PROPATH**.

## Global configurations using the contributed extension

The ABL base plugin adds the `abl` extension, which allows you to configure numerous ABL language-specific configurations inside a DSL block. These configurations are available to all task types.

The task types can use these `abl` extension configurations by adding it to their existing configuration during runtime.

## ABLExtension

This extension is a set of ABL language-specific configurations that are available to all task types.

## Methods

Method	Description	Example
<code>propath</code>	Sets the AVM's PROPATH.	<code>propath('path1', "\${dlcHome}/path2")</code>
<code>avmOptions(Closure avmOptions)</code>	AVM options. The argument should be provided as a closure.  See <code>avmOptions</code> in the <a href="#">ABLCompile</a> section for more information.	<code>avmOptions{ tmpDir="path" tty.enabled=true }</code>
<code>dbConnection({})</code>	Adds database configurations.  See the <a href="#">ABLCompile</a> section for more information.	<code>dbConnection{ dbName = 'sports2000' connectionParameters='-S 8000' }</code>



Method	Description	Example
<code>dbConnectionReferenceId(String... refIds)</code>	Adds database task references. See the <i>id</i> property of <a href="#">DBConnection</a> for more information.	<code>dbConnectionReferenceId("id1", "id2")</code>
<code>compileOptions(Closure compileOptions)</code>	Compile options. The argument should be provided as a closure. See <code>compileOptions</code> in the <a href="#">ABLCompile</a> section for more information.	<pre>compileOptions{   multiCompile.enabled=true   outputType="json" }</pre>

## Properties

Property	Required?	Description	Example	Default value
<code>dlcHome</code>	Yes (read-only property)	The OpenEdge DLC path. This property can be set in the following ways:  <ol style="list-style-type: none"> <li>1. Set <i>DLC</i> as a system property (<code>-DDL= &lt;dlcPath&gt;</code> set on the command line)</li> <li>2. Set <i>DLC</i> as a Gradle property (<code>DLC= &lt;dlcPath&gt;</code> set in <code>gradle.properties</code> file )</li> <li>3. Set <i>DLC</i> as an environment variable</li> </ol> The order of priority is 1 > 2 > 3.	None	None
<code>openedgeVersion</code>	No (read-only property)	Specifies the OpenEdge version. This can be used for logging. This is a read-only property.	None	None
<code>rcodeDir</code>	No	Sets the directory where the r-code will be generated.	<code>rcodeDir="rcode2"</code> or <code>rcodeDir="path"</code>	<code>\${buildDir}/rcode</code>

Property	Required?	Description	Example	Default value
<code>propath</code>	No	Sets the AVM's PROPATH.	<code>propath = files('path1', "\${dlcHome}/path2")</code>	None
<code>wrkDir</code>	No	Sets the working directory.	<code>wrkDir = 'path1'</code>	<code>"\${projectDir}"</code> .
<code>avmOptions{}</code>	No	AVM options.  Use the method <code>avmOptions(Closure closure)</code> to set the properties.	None	None
<code>dbConnections</code>	No	Database connection details.  To add a database connection, use methods, <code>dbConnection(Closure connection)</code> or <code>dbConnectionReferenceId(String refIds)</code> .	None	None
<code>compileOptions{}</code>	No	Compile options.  See the <a href="#">ABLCompile</a> section for more information.	None	None

### Sample code snippet

The following code snippet is an example using `ABLExtension`:

```
abl{
    rcodeDir='${buildDir}/myRcode'
    wrkDir = "."
    propath("${dlcHome}/tty/")
    avmOptions{
        tmpDir="${buildDir}"
        tty.enabled=true
    }
    compileOptions{
        multiCompile.enabled=true
        outputType="json"
    }
    dbConnection{
        dbName = 'sports2000'
        connectionParameters='-S 8000'
    }
    dbConnectionReferenceId("id1", "id2")
}
```

## Transform

[Transform](#) is a Gradle concept that is used to transform or modify dependencies. OpenEdge DevOps Framework (OEDF) has added a transform for extracting PL file dependencies. For more information about the OEDF transform, see [ExtractPL](#).

## ABL plugin

The `ABL plugin` expands the capabilities of the `ABL Base plugin` to provide various default tasks and conventions. The `ABL plugin` extends the Gradle's [Base plugin](#), and uses its lifecycle tasks to group the default tasks.

### Project requirement

The `ABL plugin` depends on the Progress Developer Studio project structure and is recommended for the Progress Developer Studio users. The mandatory files are:

- `.propath` file—Described in the [Propath file](#).
- `build.config` file—Described in the [Build config file](#).

The `ABL plugin` provides various tasks required to build an ABL project (such as task to compile ABL sources) by default, based on the project's `propath` and properties, enabling. This enables you to build a project with minimum effort and enforces the best practices for your build.

These predefined tasks and the necessary parameters required for these tasks (for example, compiling ABL files) can be configured in the `build.config` file that is part of each Developer Studio project. This way you do not have to maintain a separate configuration for development environments and build environments.

Users who do not have a Progress Developer Studio for OenEdge project but prefer to use predefined tasks can use the `ABL plugin`, provided the users add and configure the required `build.config` and `.propath` files for their source artifacts. For more information, see the [Build config file](#) and [Propath file](#).

### Usage

To use the `ABL plugin`, include the following in your build script `build.gradle` file.

```
plugins {  
    id "progress.openedge.abl " version <version>  
}
```

## Propath file

This file is part of a Developer Studio project and contains details of the project's source paths and propaths.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<propath version="12.3" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="schema/propath.xsd">
  <propathentry env="all" kind="src" path="@{ROOT}\src" platform="All"/>
  <propathentry env="gui" kind="con" path="com.openedge.pdt.text.GUI_LIBRARIES"
platform="All"/>

  <propathentry env="tty" kind="con" path="com.openedge.pdt.text.TTY_LIBRARIES"
platform="All"/>
  <propathentry env="all" kind="con" path="com.openedge.pdt.text.DLC_PATHS"
platform="All"/>

  <propathentry env="all" kind="lib" path="@{ROOT}\lib\sports2000.pl" platform="All"/>

</propath>
```

---

**Note:** Refer to [PROPATH properties page](#) topic for more information about the propath file and its attributes.

---

The propath entries with `kind=src` denotes the source path for compilation. A compile task is created for each of these source entries. For more information, see [Tasks](#).

The default location of this file is `<project-root-location>/.propath`. Though this file you can configure, if required, a custom propath file path by setting the property `CUSTOM_PROPATH_FILE`. To configure this property, check the [Configure a property](#) section in the [Prerequisites](#) topic.

## Build config file

The `build.config` file contains all the project configurations required to build a project. This file allows you to move your project along with its property settings so that your development and integration environment use the same configuration and you do not need to set up the configurations again.

---

**Note:**

- You can use the `CUSTOM_BUILD_CONFIG_FILE` to configure and provide a custom `build.config` file. For more information about configuring this property, see [Configure a property](#) in the [Prerequisites](#) topic.
  - The `build.config` file had a different name in a previous release, see [FAQ about changes to PDSOE](#) for more information.
- 

This topic describes the parameters in the `build.config` file.

**Note:** Progress recommends that you do not modify the `build.config` file manually within Progress Developer Studio. All the parameters can be updated from the Progress Developer Studio user interface (**Project Properties**) with the exception of the following:

- `streamIO.enabled`
- `languages.list`
- `textSegGrow`
- `minSize.enabled`

You can update the `build.config` file manually to build the project outside of Progress Developer Studio. The database and `compilableFileExtensions` parameters are not used with Progress Developer Studio.

For file portability, Progress recommends that you use variables instead of an absolute path for the following attributes:

- `'buildDir'`
- `'oeide.xrefXmlDir'`
- `'oeide.preCompileCallbackRoutine'`
- `'avm.wrkDir'`
- `'avm.avmOptions.tmpDir'`
- `'avm.avmOptions.assembliesDir'`

The following default variables are supported:

- **ROOT**—Root location of the project.
- **WRKDIR**—Working directory; set during installation

**Note:** **system properties**, **system environment variables**, and **Eclipse variables** are also supported.

The variable should be used as "`${variable-name}`", for example, `buildDir="${ROOT}/myBuildDir"`.

## Global parameters

Parameters	Default value	Description
<code>buildDir=''</code>	—	Specifies the project build directory.  This directory contains the saved r-code files. If this field is blank, then the r-code files are saved in the same directory as the source files.

## oeide parameters

The following table contains the IDE-specific configurations that are only used by Progress Developer Studio for OpenEdge.

Parameter	Default value	Description
<code>useSharedAVM= ''</code>	<code>false</code>	<p>This parameter specifies whether to use a shared ABL Virtual Machine (AVM) when building projects in Progress Developer Studio.</p> <p>This parameter uses a Boolean value to specify whether the project uses a shared or a dedicated AVM.</p>
<code>oeideEvents= ''</code>	<code>true</code>	<p>This parameter uses a Boolean value to enable or disable oeide events. This parameter enables you to write procedures that use the ABL SUBSCRIBE statement to capture and respond to those events.</p>
<code>useGlobalToolboxVD= ''</code>	<code>false</code>	<p>This parameter uses a Boolean value to enable or disable the global Visual Designer toolbox.</p>
<code>addDefaultParams= ''</code>	<code>true</code>	<p>This parameter specifies whether to use the default startup parameters when starting the AVM.</p> <p>This parameter uses a Boolean value to enable or disable the passing of default AVM parameters.</p>
<code>hideTTYConsole= ''</code>	<code>true</code>	<p>This parameter uses a Boolean value to enable or disable the project-specific run-time console.</p>
<code>useProjectCompilerSettings= ''</code>	<code>false</code>	<p>This parameter uses a Boolean value to enable or disable the use of project-specific <code>strictOptions</code> under <code>compile</code>.</p>
<code>saveRCode= ''</code>	<code>true</code>	<p>This parameter uses a Boolean value to enable or disable persistent r-code on compiling sources.</p>
<code>xrefXmlDir= ''</code>	—	<p>This parameter specifies the output value of the XREF-XML option when <code>xrefXml.enabled= 'true'</code>.</p> <p>This parameter uses a string value to specify the path where cross-reference information is saved in an XML file.</p>

Parameter	Default value	Description
<code>preCompileCallbackRoutine= ''</code>	—	This parameter uses a string value to specify the path to an ABL file that is run before compilation.  For example, this option can used to set <code>SECURITY-POLICY: XCODE-SESSION-KEY</code> .

### AVM parameters

Parameter	Default value	Description
<code>wrkDir= ''</code>	The project's root directory	This parameter uses a string value to specify the working directory in which the AVM starts.

### AVM option parameters

The following table lists the AVM option parameters (defined in the `build.config` file as `avmOptions`).

Parameter	Default value	Description
<code>tmpDir= ''</code>	The DLC working directory	This parameter specifies the temporary directory for the AVM run time (equivalent to the <code>-T</code> startup parameter option).  This parameter uses a string value to specify the path.
<code>tty.enabled= ''</code>	false	Flag to use <code>_progres</code> or <code>prowin</code> executables. For <code>_progres</code> , the value is <code>true</code> ; otherwise, it is <code>false</code> .  This parameter uses a Boolean value.
<code>startupParameters= ''</code>	—	This parameter uses a string value to specify the startup parameters as a string.
<code>assembliesDir= ''</code>	—	Specifies the location of the <code>Assemblies</code> directory (equivalent to the <code>-assemblies</code> startup parameter option).  This parameter uses a string value.

### Compile parameters

Parameter	Default value	Description
<code>compilableFileExtensions= ''</code>	<code>p, w, cls, pgen, html, htm</code>	This parameter allows you to set the compilable file extensions.

### Compile option parameters

The following table lists the Compile option parameters (defined in the `build.config` file as `compileOptions`).



Parameter	Default value	Description
<code>multiCompile.enabled= ''</code>	<code>false</code>	To enable the <code>COMPILER: MULTI-COMPILE</code> attribute, set the value to <code>true</code> .
<code>xcodeKey= ''</code>	—	<code>XCODE</code> option in the <code>COMPILE</code> statement. Specify the expression as a string.
<code>xrefXml.enabled= ''</code>	<code>false</code>	<code>XREF-XML</code> option in the <code>COMPILE</code> statement. To enable this option, set the value to <code>true</code> . Specify the expression as a Boolean value.
<code>streamIO.enabled= ''</code>	<code>false</code>	<code>STREAM-IO</code> option in the <code>COMPILE</code> statement. To enable this option, set the value to <code>true</code> . Specify the expression as a Boolean value.
<code>minSize.enabled= ''</code>	<code>false</code>	<code>MIN-SIZE</code> option in the <code>COMPILE</code> statement. To enable this option, set the value to <code>true</code> . Specify the expression as a Boolean value.
<code>attrSpace.enabled= ''</code>	<code>false</code>	To enable the <code>ATTR-SPACE</code> option in the <code>COMPILE</code> statement, set the value to <code>true</code> .

### Strict option parameters

The following table lists the Strict option parameters (defined in the `build.config` file as `strictOptions`).

Parameter	Default value	Description
<code>requireFullNames=''</code>	ignore	This parameter specifies the <code>OPTIONS</code> <code>require-full-names</code> in the <code>COMPILE</code> statement. Value can be ignore, warning, or error.
<code>requireFieldQualifiers=''</code>	ignore	This parameter specifies the <code>OPTIONS</code> <code>require-field-qualifiers</code> in the <code>COMPILE</code> statement. Value can be ignore, warning, or error.
<code>requireFullKeywords=''</code>	ignore	This parameter specifies the <code>OPTIONS</code> <code>require-full-keywords</code> in the <code>COMPILE</code> statement. Value can be ignore, warning, or error.
<code>requireReturnValues=''</code>	ignore	This parameter specifies the <code>OPTIONS</code> <code>require-return-values</code> in the <code>COMPILE</code> statement. Value can be ignore, warning, or error.

## Languages parameters

Parameter	Default value	Description
<code>list=''</code>	—	Comma-separated list of language segments to include in the compiled r-code.
<code>textSegGrow=''</code>	—	This parameter specifies the <code>TEXT-SEG-GROW</code> option. The value must be an integer. Supported only when a language list is provided.

## Tasks

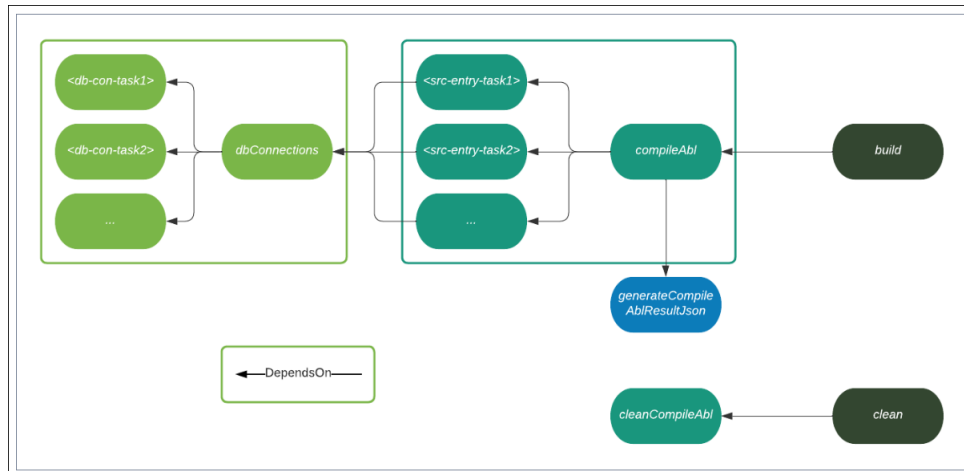
To build an ABL project, the `ABL` plugin provides various tasks. For example, a task to compile ABL sources by default, based on the project's `propath` (`.propath` file) and properties (`build.config` file).

You can refer to the [Understand the build process of an ABL project](#) topic to understand in greater details about various stages of a project build.

By default, a project consists of the following tasks:

- Build
- compileAbl
- dbConnections
- clean
- cleanTaskName

This image shows the relationship between the tasks:



## build (lifecycle task)

This task is used to build a project and relies on the `compileAbl` task. Currently, the build task only compiles the project and generates the `r-code`. In future, it will do all other necessary things<sup>x</sup> needed for building the project.

## compileAbl

The `compileAbl` task compiles an ABL project. It compiles the ABL sources for all the sourcepath entries defined in the `propath` file. This task has `dependsOn` dependency on the `<src-entry-tasks>`.

In a scenario, where you run the `compileAbl` task with more than one `<src-entry-tasks>` and if one of them fails due to a compilation error or otherwise, then the other `<src-entry-tasks>` will fail to run. This is the default Gradle behavior where if one dependent task fails the the other tasks are not run.

---

**Note:** Use `--continue` to get complete feedback on compilations errors for all the `src-entries`. For example:

```
$gradle compileAbl --continue
```

---

## <src-entry-tasks>

A task (of type `ABLCompile`) is created to compile ABL sources for each sourcepath entry (`kind=src`) defined in the `propath` file.

The name of the task follows the convention `compileAbl-root-subpath-path-from-root`. For example for a sourcepath, `@{ROOT}/src`, the task name would look like `compileAbl-root-src`.

When compiling the sources, you may prefer to configure these tasks for various options as described in the [Compile source code](#) section.

Follow these steps to configure the task:

- AVM configurations
  - Use [Propath file](#) or refer to [Dependency Management](#) to provide **PROPATH** specific dependencies.
  - Use [Build Config file](#) to configure database connection details. A `dbConnections` task is created to hold these connection details. All these `<src-entry-tasks>` have `dependsOn` dependency on the `dbConnections` task.
  - Use [Build Config file](#) to configure AVM parameters.
- Compile configurations
  - Use [Build Config file](#) to configure various compile configurations.

---

**Note:** You can also use the global configurations, described in the [ABL base plugin](#) section, to configure various properties of the tasks. However, you might not always have a requirement to do so.

---

### **generateCompileAblResultJson**

Generates a unified result of the JSON output of different tasks. This output is used when the property `outputType` is set in `compileOptions`.

### **dbConnections**

Holds ABL database connection details that are defined in the config file. A task (of type `DBConnection`) is created for each connection entry provided in the [Build Config file](#).

### **clean**

Cleans the project by removing the build output. Run this task first if you want to re-build the project in a clean environment.

### **cleanTaskName**

Deletes files created by the specified task. For example, `cleanCompileAbl` deletes the `r-code` files and any other output files generated by the `compileAbl` task.

## **Dependency Management**

This section describes how to manage dependencies when building an ABL project using the ABL plugin. Refer to the [Manage dependencies](#) section to understand about various kinds of dependencies that you might want to handle.

You can handle PROPATH specific dependencies in the following ways:

1. Using Propath file (this approach can only handle dependencies that are present locally)—Refer to the [Propath file](#) section for more details.
2. Using dependency configurations—The ABL plugin has contributed to Gradle configurations using which you can manage your PL file dependencies both from local and from remote repositories.

Managing dependencies using dependency configurations is a two-step process— declaring repositories, and declaring dependencies.

## Declaring Repositories

OEDF supports all types of repositories as supported by Gradle such as Maven, Ivy, and local (flat) directories. For more information on how to configure repositories, see [Gradle documentation](#).

### Example

```
//declare repositories

repositories {

    maven {

        url = '<url-to-maven-repository>'

        metadataSources {

            artifact()//needed to download artifacts (like .pl file) without a pom file in the
            repository

        }

    }

}
```

## Declaring Dependencies

Every dependency declared for a project applies to a specific scope. For example, some dependencies should be used for compiling source code whereas others may only need to be available for testing. Gradle represents the scope of a dependency with the help of a configuration.

OpenEdge DevOps Framework introduces different configurations for various types of dependencies for ABL projects. (Currently, only default tasks for compilation have been added, so naturally, the scope of these configurations is limited to compilation).

These configurations support both Module dependencies and File dependencies. For more information, see [Gradle documentation](#). You can also configure the dependencies further as supported by Gradle. For example, to enable whether Gradle should always check for a change in the remote repository.

See [Dependency configurations to handle PL dependencies](#) for the list of configurations that should be supported in OpenEdge DevOps Framework for ABL.

## Dependency configurations to handle PL dependencies

This sections describes the list of configurations supported in OpenEdge DevOps Framework for ABL.

### implementationAbl

The *implementation only* dependencies use at both compile-time and runtime.

#### Example

```
//define the dependencies

dependencies {

    implementationAbl(<group>:<module-name>:<version>:<classifier>@pl')

}
```

### compileOnlyAbl

The dependencies used only for compilation (not used at runtime).

**Example**

```
//define the dependencies

dependencies {

    compileOnlyAbl(<group>:<module-name>:<version>:<classifier>@pl')

}
```

**compilePropath**

Extends `compileOnlyAbl` and `implementationAbl` used when compiling sources. Used by the `compileAbl` (`<src-entry-tasks>`) task.

**Manage dependencies and propaths from .propath**

The propaths from `.propath` file are added first, followed by the dependencies coming from the dependency configurations. Priority is given to the dependencies coming from the `.propath` file.

**configurationNameSource**

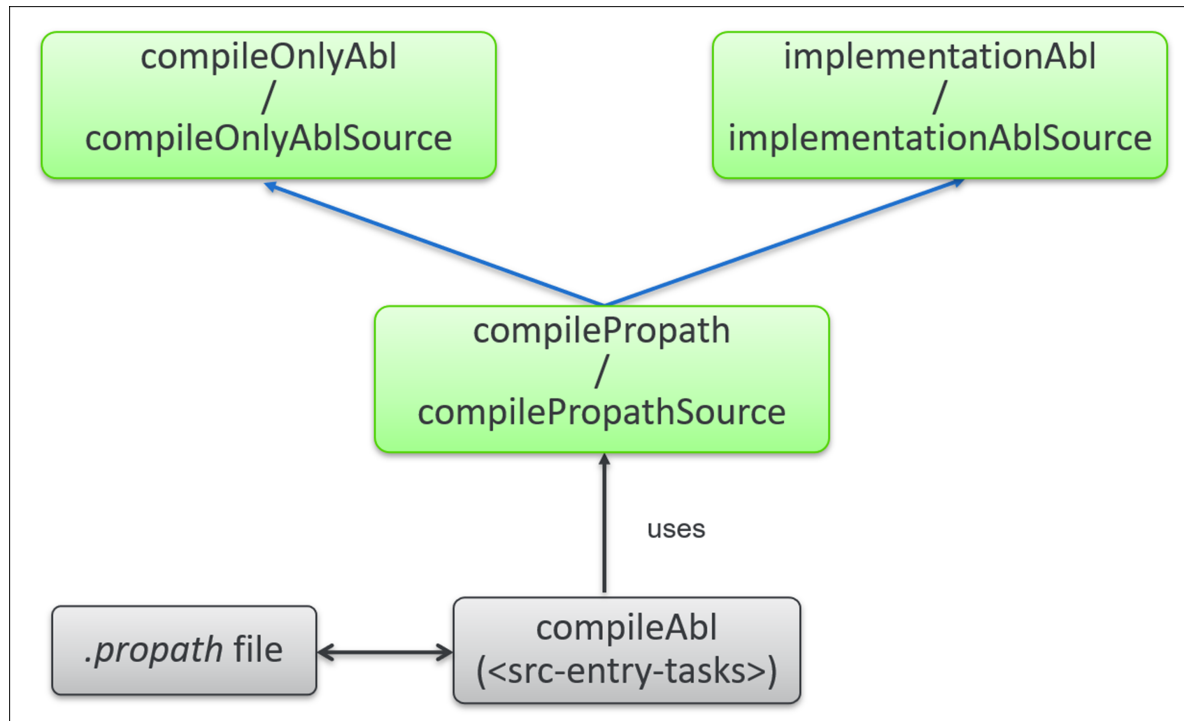
For each of the above dependency configurations, one dependency configuration is added to handle PL file dependencies with sources. See [Procedure Library \(PL\) dependencies in PROPATH](#) for more information.

Such PL file dependencies are treated similar to their counterparts, except that the PL file dependencies defined by this configuration are extracted using the `ExtractPLTransform` transform and then added to the `.propath`.

**Example**

- `implementationAblSource` for `implementationAbl`.
- `compileOnlyAblSource` for `compileOnlyAbl`.
- `compilePropathSource` for `compilePropath`.

This illustration shows the relations between dependency configuration and tasks.



## Sample Example

```

plugins {
    id "progress.openedge.abl" version "<version>"
}

//declare repositories
repositories {
    maven {
        url = '<maven-repo-url>'
        metadataSources {
            artifact() //needed to download artifacts without pom file
        }
    }
}

//define the dependencies
dependencies {
    implementationAbl(<group>:<module-name>:<version>:<classifier>@pl')
    implementationAblSource(<group>:<module-name>:<version>:<classifier>@pl')
    compileOnlyAbl(<group>:<module-name>:<version>:<classifier>@pl')
    compileOnlyAblSource(<group>:<module-name>:<version>:<classifier>@pl')
}

```

In this example, the dependencies defined by `implementationAbl` and `compileOnlyAbl` are added to the `.propath` directly as PL files, whereas dependencies defined by `implementationAblSource` and `compileOnlyAblSource` are first extracted and then the location of the extracted directory is added to the `.propath`.

## Build packages for PAS for OpenEdge application

Progress Application Server (PAS) for OpenEdge supports various package artifacts like `oear`, `oewar`, `oesvczip`, `oeds`, and `paar`, that can be deployed to a PAS for OpenEdge server instance. OpenEdge DevOps Framework (OEDF) is a unified build solution for ABL and PAS for OpenEdge and enables you to generate the PAS for OpenEdge packages that can be used for testing and deployment.

A PAS for OpenEdge application consists of an instance layer, ABL application layer, one or more ABL web application layer, and one or more ABL service layer. Each of these layer has its own package type that is used for deployment on a PAS for OpenEdge server instance. OEDF provides [task types](#) using Gradle plugins to create the various packages that can be deployed on a PAS for OpenEdge instance.

The following artifacts are created for a PAS for OpenEdge application:

- **Instance**—Instances are a standard Apache Tomcat feature and independently run copies of the PAS for OpenEdge installation on a PAS for OpenEdge instance. You can deploy an instance by compressing the instance folder, copying it to the production machine, and then registering and starting the instance.

For more information about instance, see "Instances" in the *Learn about Progress Application Server for OpenEdge*.

Using OEDF, you can package a PAS for OpenEdge instance as any generic archive such as ZIP, using generic the `zip` task type of Gradle and then register it with PAS for OpenEdge server instance.

- **ABL Application**—A PAS for OpenEdge instance and the ABL applications that it hosts have a defined structure with specific locations for different components. An ABL application is the smallest unit of packaging for business application logic that can run independently on a PAS for OpenEdge instance. All PAS for OpenEdge instances must have at least one ABL application deployed on a PAS for OpenEdge instance.

For more information about ABL application, see "ABL applications" in *Learn about Progress Application Server for OpenEdge*.

OEDF provides [Oear](#) on page 44 task type to create an ABL application package for deployment on a PAS for OpenEdge instance.

- **ABL Web Application**—The ABL web applications deployed to a PAS for OpenEdge instance are generated as Web Application Archive (WAR) files. An ABL application contains at least one ABL web application derived from the `oeabl.war` application. An ABL web application exposes ABL application logic using HTTP, so it can be called with a URI and with HTTP methods.

For more information about ABL web application, see "Web applications" in *Learn about Progress Application Server for OpenEdge*.

OEDF provides [OEWAr](#) on page 49 task type to create an ABL web application level task package.

- **ABL Service**—An ABL service is a set of business logic that can be accessed through a URL path.

For more information about ABL web application, see "Transports and services" in *Learn about Progress Application Server for OpenEdge*, and "ABL Service Deployment" in *Manage Progress Application Server (PAS) for OpenEdge*.

OEDF provides [Oeds](#) on page 56 task type for creating ABL service level package, [OESvcZip](#) on page 53 task type for creating incremental ZIP files for REST and WEB service package, and [Paar](#) on page 60 task type for creating package for the REST service, for deployment on a PAS for OpenEdge instance.

## Migrate from Apache Ant

Many ABL applications make use of [Apache Ant](#) and [PCT](#). [Apache Ant](#) is a build automation tool that uses XML and Java, and it was designed to make build processes simpler and more portable. Ant uses targets and tasks to compile, test, and run applications. It is primarily used for Java applications, but can be used for other types of applications as well. [PCT](#) is an open source set of [Apache Ant](#) tasks and plugins created by Riverside Software that automate work in OpenEdge. PCT is a large ecosystem of plugins that provides common build steps for automation and CI/CD.



## Why migrate from Ant to Gradle?

Migrating from Ant to Gradle has the following benefits:

- Gradle uses a domain-specific language, which allows users to add logic to the build pipeline.
- Gradle has reusable tasks and can reuse other parts of a build pipeline, which makes the build script less verbose, more readable, and easier to maintain.
- Gradle provides smarter dependency management than Ant.
- Gradle make builds modular, which allows larger projects to be built from interdependent parts. You can even build multiple projects from the same pool of parts.

## Use `ant.importBuild`

The Gradle task `ant.ImportBuild` is a straightforward and quick way to migrate your build processes from PCT or Ant to Gradle. This task converts Ant targets in your Ant scripts into Gradle tasks. To use `ant.ImportBuild`:

1. Add the following to your `build.gradle` file in your project:

```
ant.ImportBuild 'build.xml'
```

This assumes `build.xml` contains your previous Ant tasks.

2. After saving your `build.gradle` file, run your previous Ant tasks using the Gradle wrapper:

```
gradlew ant target
```

Using `ant.ImportBuild` works quite well as an initial phase of migrating to Gradle, but it is not a full migration because you still need to maintain your existing Ant scripts.

