Progress®



Modernize OpenEdge Applications

OpenEdge®

Copyright

© 2021 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Defrag This, Deliver More Than Expected, DevReach (and design), Icenium, Inspec, Ipswitch, iMacros, Kendo UI, Kinvey, MessageWay, MOVEit, NativeChat, NativeScript, OpenEdge, Powered by Chef, Powered by Progress, Progress, Progress Software Developers Network, SequeLink, Sitefinity (and Design), Sitefinity, Sitefinity (and design), SpeedScript, Stylus Studio, Stylized Design (Arrow/3D Box logo), Styleized Design (C Chef logo), Stylized Design of Samurai, TeamPulse, Telerik, Telerik (and design), Test Studio, WebSpeed, WhatsConfigured, WhatsConnected, WhatsUp, and WS_FTP are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries.

Analytics360, AppServer, BusinessEdge, Chef Automate, Chef Compliance, Chef Desktop, Chef Habitat, Chef WorkStation, Corticon.js, Corticon Rules, Data Access, DataDirect Autonomous REST Connector, DataDirect Spy, DevCraft, Fiddler, Fiddler Everywhere, FiddlerCap, FiddlerCore, FiddlerScript, Hybrid Data Pipeline, iMail, JustAssembly, JustDecompile, JustMock, KendoReact, NativeScript Sidekick, OpenAccess, PASOE, Pro2, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Push Jobs, SafeSpaceVR, Sitefinity Cloud, Sitefinity CMS, Sitefinity Digital Experience Cloud, Sitefinity Feather, Sitefinity Insight, Sitefinity Thunder, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Supermarket, SupportLink, Unite UX, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

July 2021

Product version: Progress OpenEdge 12.4

Preface

Purpose

This guide provides an overview of how to modernize a legacy OpenEdge application. It describes the key components in OpenEdge 12 that enable modernization and lists the main steps involved in transforming a legacy OpenEdge application into a modern, cloud-ready, business application that follows OpenEdge Application Architecture (OEAA) principles.

Audience

This guide is aimed at OpenEdge solution architects and application developers who want to learn about modernization in OpenEdge.

Organization

This guide consists of the following parts:

- Learn about modernization in OpenEdge on page 7: Provides an overview of modernization in OpenEdge
- Analyze existing architecture on page 15: Describes the set of actions that you may want to perform to analyze your modernization requirements
- OpenEdge application design on page 17: Describes the modernization design process

Documentation conventions

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

Get Started

For details, see the following topics:

- Learn about modernization in OpenEdge
- Modern OpenEdge deployment architecture
- Modernization steps

Learn about modernization in OpenEdge

Application modernization is the transformation of a legacy application into a modern, cloud-ready business application that supports strategies like Software as a Service (SaaS), continuous integration and continuous development (CI/CD), and API-first development. If you are an OpenEdge solution architect or developer, you can jump-start your modernization process with OpenEdge 12—the most secure and performant version of OpenEdge ever released.

Why modernize?

Over the years, OpenEdge has helped manage mission-critical aspects of businesses across a range of domains—from ERP to mortgage loans to insurance claims to payroll. Many enterprises continue to use OpenEdge with little to no changes in their applications because applications built on OpenEdge are robust and require little maintenance.

However, with the rise of cloud computing, the technological landscape has changed dramatically. Customers increasingly want their applications to run online, be accessible on a variety of devices, and be available all the time. OpenEdge 12 comes with a set of features that are designed to help you adapt to these changes.

Before you get into the details of modernizing a legacy OpenEdge application, take some time to understand the key characteristics of a modern application, the OpenEdge features that enable modernization, and the typical modernization workflow.

Characteristics of a modern application

Modern applications are typically:

- API-centric—APIs enable extensibility and scalability. By exposing parts of your application through individual APIs, you allow for easier integration with internal and external systems.
- UX-focused—When everything is an API, front-end developers are empowered to design better user experiences.
- Cloud-enabled—Applications are delivered and consumed over the cloud. The application architecture is modular, distributed, and designed with an emphasis on scalability and performance.

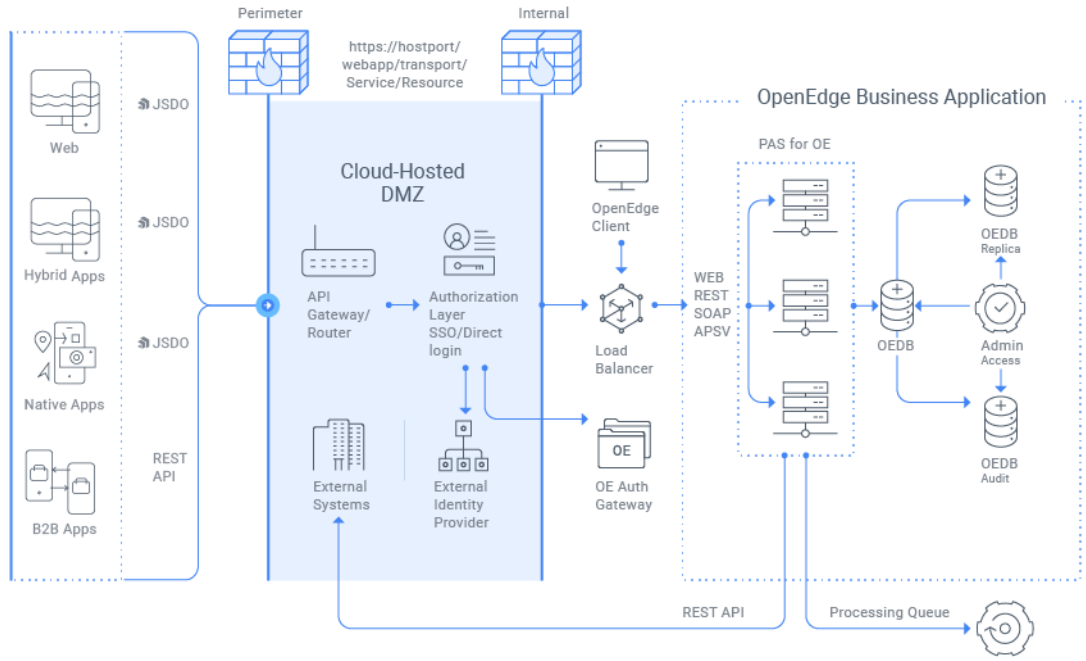
OpenEdge features that enable modernization

OpenEdge provides a wide set of tools that enable you to design and develop modern business applications. The key features that enable modernization include:

- Progress Application Server for OpenEdge (PAS for OpenEdge)—Based on Apache Tomcat, PAS for OpenEdge provides a modern web server that is tailored to host ABL web applications.
- ABL web handlers—ABL web handlers enable you to easily wrap your ABL code in RESTful APIs. All you do is write simple service interface procedures and let the ABL web handler handle the difficult task of transforming HTTP messages into ABL and vice versa.
- Progress Data Object Services—A Data Object Service is a specific type of RESTful ABL service that enables you to integrate your ABL business logic with JavaScript Data Object (JSDO) client applications.

Additionally, in OpenEdge 12, you can run PAS for OpenEdge in a Docker container, giving you the ability to set up CI/CD pipelines for faster development and deployment.

Modern OpenEdge deployment architecture



A modern OpenEdge deployment architecture is composed of several parts:

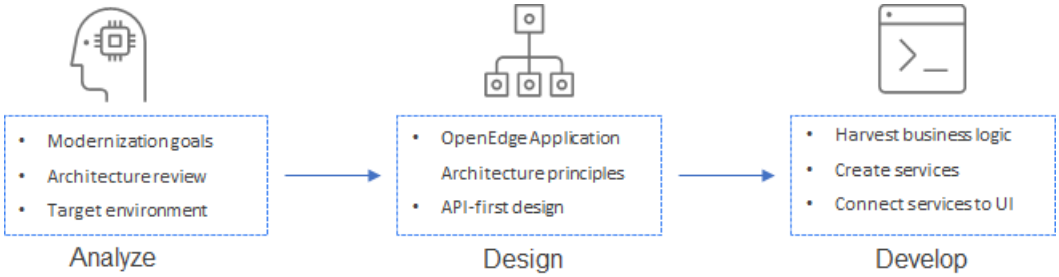
- **OpenEdge Business Application**—The core of the architecture is the ABL business application. The ABL business application runs in a PAS for OpenEdge instance and is connected to an OpenEdge database. The business logic contained within the application is exposed as a set of ABL services.
- **ABL services**—ABL services are an important feature of a modern OpenEdge environment.
  - Each service is an API that can be consumed by a client application.
  - You can create different types of ABL services such as REST or SOAP.
  - Each service is bound to a PAS for OpenEdge transport that transforms HTTP messages into ABL. The WEB transport, which is the recommended way to set up a RESTful ABL service, uses ABL web handlers, that are highly customizable and written in ABL. Other transports include SOAP (for SOAP communication), REST (an alternative to the WEB transport for RESTful services), and APSV (for ABL clients).
  - Services are packaged in an ABL web application that can then be deployed to an ABL application in a PAS for OpenEdge instance.
- **Cloud-hosted DMZ**—After your business logic is exposed through services, you can guard the services using a cloud-based demilitarized zone (DMZ) that may include the following components.
  - An authentication and authorization layer managed by the **OpenEdge Authentication Gateway** or an external identity provider like Okta, OneLogin, etc, that use security mechanisms like SAML and OAuth.

    OpenEdge Authentication Gateway is an identity management system that leverages PAS for OpenEdge as the underlying technology. It uses a Secure Token Service (STS) to confirm user legitimacy and assigns a standard, strongly encrypted ABL Client-Principal token to authenticated clients. The Client-Principal token is then used to maintain a chain of trust between the client and an OpenEdge application.

- An API gateway that serves as an entry point for your APIs. You can use an API gateway to create a single API that is used by client applications. When a client application calls this API, the API gateway server invokes your ABL services and aggregates the results into a single response that is returned to the client application.
- Load balancing mechanisms that distribute client requests across multiple PAS for OpenEdge server instances. There are multiple load balancing options that you can choose from such as Apache Tomcat, Apache HTTP, and Amazon Elastic Load Balancing.

- **Front-end client applications**—Front-end client applications include web, mobile, and desktop clients. The user interface is typically built using HTML, CSS, and JavaScript libraries such as JSDO (JavaScript Data Objects—an OpenEdge client-side framework designed to work with OpenEdge Data Object services), Kendo UI, AngularJS, Vue, and React. When users interact with components in the user interface, calls are made to the API gateways, identity providers, and so on. After the calls pass through those systems, the calls arrive at the boundaries of PAS for OpenEdge, where they are processed by ABL services.

Modernization steps



Even though the exact set of steps to modernize your OpenEdge application depends on your specific requirements, the modernization process typically involves the following phases:

1. Analyze
   - Define your modernization goals.
   - Analyze your existing architecture.
   - Identify opportunities for reuse.
   - Identify modernization requirements.
2. Design
   - Create a modern design based on OpenEdge Application Architecture principles.
   - Incorporate continuous availability in your deployment design.
3. Develop
   - Set up a Progress Developer Studio for OpenEdge environment.
   - Decouple ABL business logic from the UI code.
   - Develop ABL services for a web application.
   - Connect the ABL services to the UI layer.

This is not an exhaustive list. Its purpose is to illustrate the types of activities that you may need to perform in your environment. Keep in mind that it is more efficient and cost effective to make small incremental changes, rather than overhauling your systems.

II

Analyze

For details, see the following topics:

- Define modernization goals
- Analyze existing architecture
- Identify modernization requirements

Define modernization goals

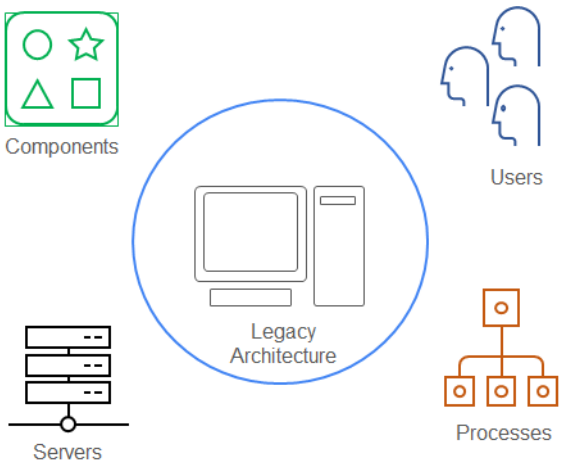
The modernization process begins with an analysis of your requirements. During this phase you define modernization goals, review your legacy architecture, identify opportunities for reuse, and finally, determine requirements for a modern OpenEdge solution.



The first step is to determine your business goals and identify the OpenEdge features that will enable you to reach those goals. Consider the factors that extend beyond your application, for example:

- What does a successful modernization process look like to you? It may be useful to start by describing the end result, for instance: "By the end of this modernization process, I want to have increased application uptime, decoupled operations, a service-oriented architecture, and improved security". By clearly articulating your business goals, you can focus your effort on the aspects of modernization that are most important to your business.
- What OpenEdge features or components will help you achieve your target environment?
  - If your key goal is increased application uptime, then take advantage of the ability to make online database changes in OpenEdge 12.
  - If your key goal is to have decoupled operations, then use an OpenEdge web handler to design and develop REST APIs for your back-end ABL business logic.
- What type of cloud environment do you need? Will your environment be hosted on a public cloud or a private cloud? Will your OpenEdge application have cloud-native characteristics such as containers and orchestration?

Analyze existing architecture



After you define your business goals, review your existing architecture and analyze your migration requirements. Here is an abbreviated list as an example:

- Users—Create a list of your users and their access requirements. Are these users accessing the application from within or outside of a firewall? What types of authentication and authorization do you need?
- Components—Inventory your existing components. For example, you may have desktop GUIs that must be replaced by browser-based UIs and database servers that must be migrated.
- Processes—Examine your current processes and determine which of them must be changed or migrated. For example, do you have batch processes or scheduled jobs that must be modified or retained?

Identify opportunities for reuse

There are many parts of your application architecture that you will want to retain, such as your core business logic and your data models. As you review your application architecture, determine which components and parts you want to keep, and which you want to discard.

Identify modernization requirements

After the completing the review of your existing architecture, you are ready to start identifying the tasks that you must perform to modernize your OpenEdge environment. Typically, these tasks include:

- Identifying the end-user workflow, the UI screens that will be presented to the user, types of client applications, the front-end technologies that will be used to build the client applications, and so on
- Designing APIs that will be consumed by the UI screens and other types of client applications
- Determining the ABL classes and procedures that will be exposed through the APIs
- Optionally, choosing frameworks that implement design patterns like the Common Component Specification
- Identifying databases and database tables that will be used by the ABL business logic

- Determining the type of database connection appropriate for your needs—shared-memory connections (where the application and database are on the same machine) or network connections (where the application and database communicate over TCP/IP)[1]
- Determining how context and application state will be managed
- Determining other factors such as containerization, load balancing, failover, monitoring, and performance

At the end of your analysis, you should have a set of actions that you need to perform. These actions typically span across different components and roles, such as a database administrator setting up a multi-tenant database and a DevOps engineer working on a CI/CD pipeline. However, because the focus of this guide is the application, the remainder describes how to design and develop a modern OpenEdge application.

[1] Although shared-memory connections (also called self-service connections) are faster in certain situations, network connections enable scalability and are the recommended way to connect to a database in a modern deployment.

III

Design

For details, see the following topics:

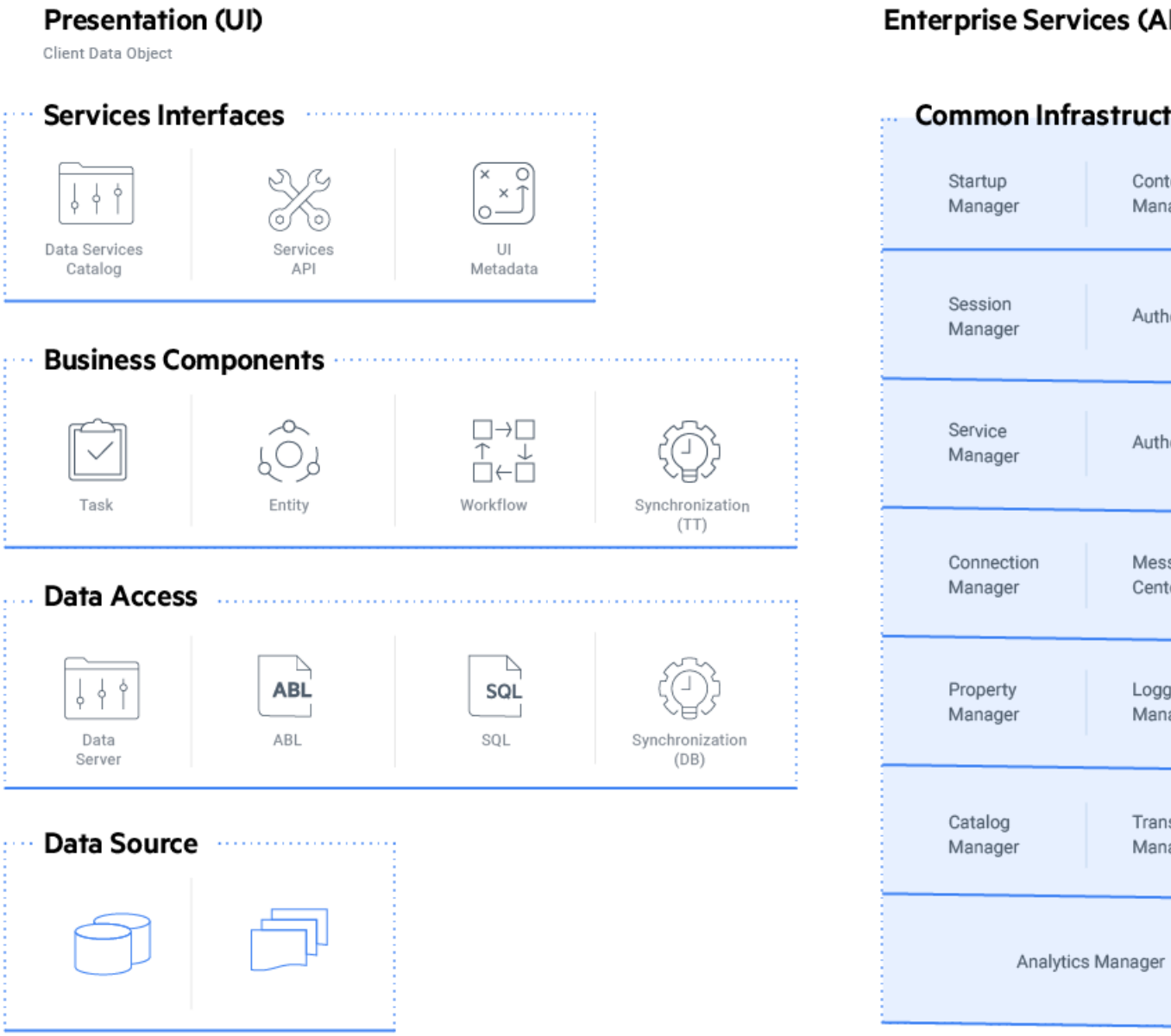
- OpenEdge application design
- OpenEdge deployment design

OpenEdge application design

A modern OpenEdge application is designed on OpenEdge Application Architecture (OEAA) principles, which prescribe isolating UI code, business logic code, and data access code into separate layers. You start the design process by identifying the APIs that will be used by client applications and then determining the underlying software components.

OpenEdge Application Architecture

OpenEdge Application Architecture (OEAA) is the recommended way to design an OpenEdge application.



OEAA consists of the following layers:

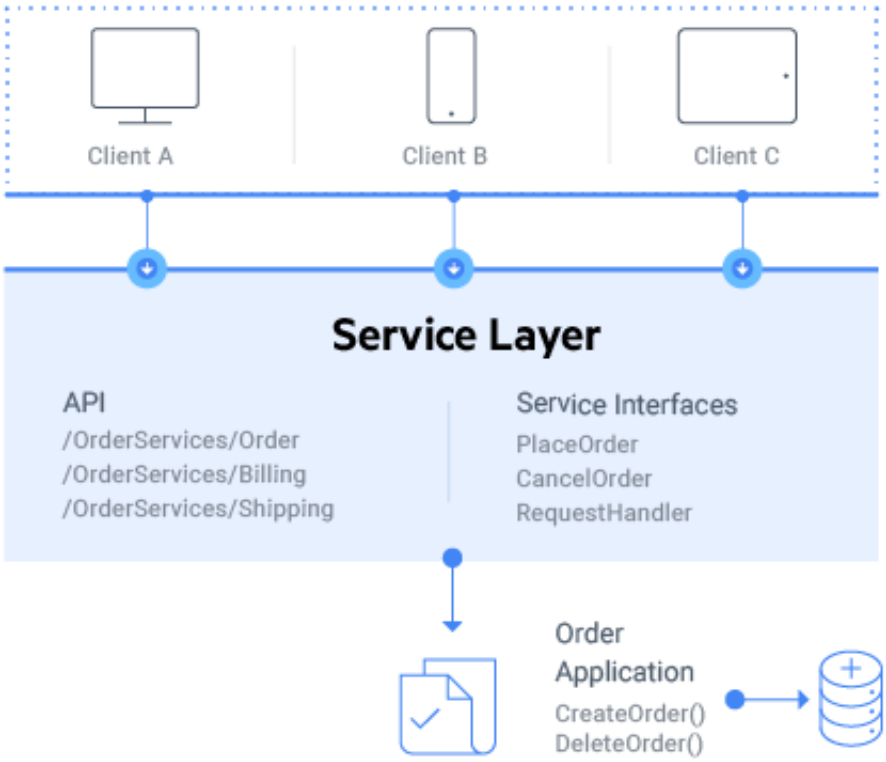
- A **Data Access** layer that manages the retrieval of data from sources such as an OpenEdge database.
- A **Business Components** layer that contains the core business logic.
- A **Service Interfaces** layer that translates requests into a format that the Business Components layer can process.
- An **Enterprise Services** layer that exposes the business logic to applications and services that exist beyond the bounds of the OpenEdge application.
- A **Presentation** layer that contains the user interface.

- A **Common Infrastructure** layer that contains framework code to support the application business logic. The recommended way to develop components in this layer is to use a framework based on the Common Component Specification.

Common Component Specification

The Common Component Specification (CCS) is a specification driven by the OpenEdge community that defines a prescriptive architecture to use when building enterprise business applications with OpenEdge. The CCS uses a service-oriented architecture (SOA) design in which application components provide services to other components through a communications protocol. To learn more about the CCS specification, see the CCS GitHub project.

API-first design



An API provides a set of functions or procedures that a client application can call on a server application. Modern applications are built API first, which means prioritizing the concerns of the client application developer. You begin by visualizing your application as a set of services that can be easily consumed by client applications. Complexities in your business logic layer are managed in the service interface and hidden from the client application. An API-first design leads to a better developer experience and enables you to add new functionality later, with minimal disruption to existing integrations.

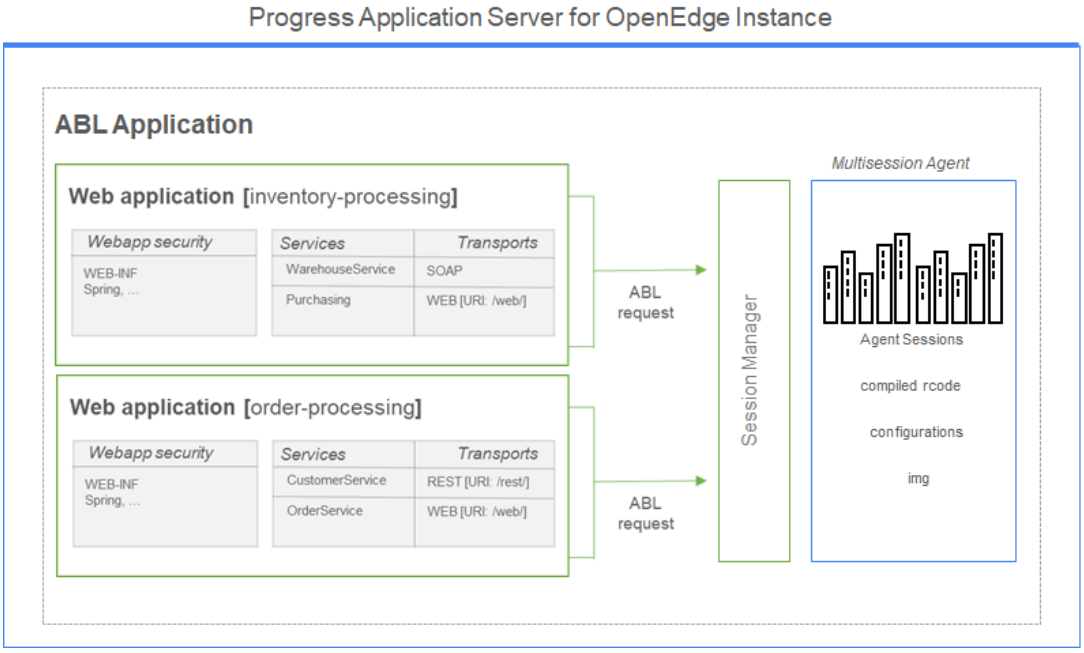
OpenEdge enables you to expose APIs to client applications through ABL services. You can think of an ABL service as having two layers—an API and a service interface. The API is the URI that a client application sends requests to. The service interface is the code that handles those requests and transforms them, if needed, so that the underlying business logic layer can process them.

To learn more about developing APIs in OpenEdge, see 'API-first development' in *Develop ABL Services*.

OpenEdge deployment design

As you design your application, you may want to consider how your application will be deployed in a distributed PAS for OpenEdge environment. The following is the hierarchy of components in a PAS for OpenEdge environment:

- You can create and run more than one instance of PAS for OpenEdge, and you can configure the instances separately. Each instance shares the executables and libraries of a common Apache Tomcat server, but each instance is a separate process that runs in a separate JVM, with its own configuration (ports, security framework, web applications).
- Each PAS for OpenEdge instance can contain multiple ABL applications.
- Each ABL application can contain multiple web applications.
- Each web application can contain multiple ABL services.
- Each ABL service is bound to a PAS for OpenEdge transport and is mapped to ABL resources (ABL classes and procedures).
- Each ABL application has its own PROPATH and a multisession agent configuration.

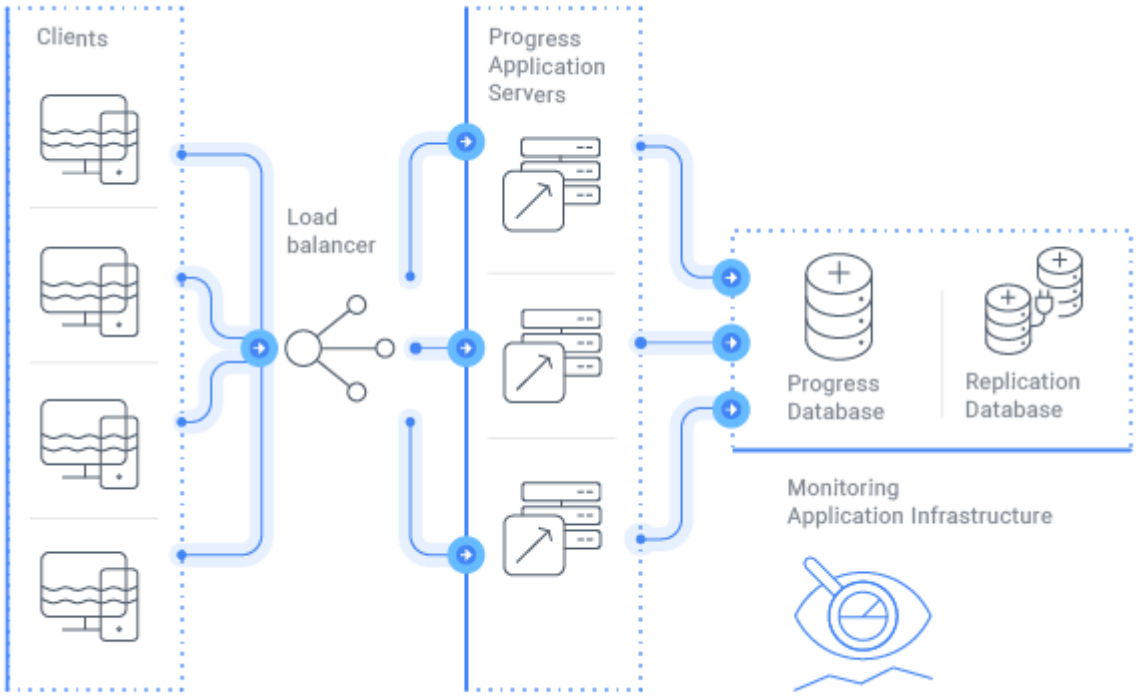


Incorporate continuous availability in your deployment design

Perhaps the most important goal of deployment design is to enable continuous availability and scalability. Continuous availability is the ability to deliver your application to end users without interruption. To achieve continuous availability, your deployment design should have:

- No single point of failure
- Use of at least two availability zones in a cloud environment
- Stateless applications
- Infrastructure as code

Deployment architecture for continuous availability



A continuously available deployment architecture is composed of several key structures:

- **Load balancer**—A full-featured load balancer is a required part of the OpenEdge continuous operations solution. The load balancer is crucial to continuous operations because it redirects traffic from any PAS for OpenEdge instance that is at risk of becoming over loaded to an instance with less traffic. OpenEdge does not have a built-in load balancer. You must choose the correct load balancer with the features that you need.
- **Progress Application Server (PAS) for OpenEdge**—PAS for OpenEdge has be deigned for scalability and to be able to work seamlessly with load balancers for continuous operation.
- **ABL and SQL applications**—Continuous operations can be achieved with applications written in ABL or SQL programing languages because they were designed to push updates and schema changes while the application is online. This eliminates the need to take your application offline for updates or maintenance.
- **OpenEdge database**—OpenEdge 12 databases are designed with continuous operation in mind, and they provide a means to update parameters, schemas, and more while they are online.
- **Data replication**—Data replication helps you avoid single points of failure by enabling a disaster recovery or a failover site. Replication ensures the availability of an application and distributes copies of data to one or more sites for horizontal scaling. Using OpenEdge database replication is the best way to make your application highly available and fault tolerant. If your production database, database machine, or network becomes unavailable, then you can quickly transition to the replicated database.
- **Monitor the system**—To have a continuously operating application, you must constantly monitor your application ecosystem, and have a system that will alert you about potential problems. Monitoring the application ecosystem allows you to observe and act on possible issues before they arise. In OpenEdge 12, this can be accomplished with the OpenEdge HealthScanner and deferred logging.

IV

Develop

For details, see the following topics:

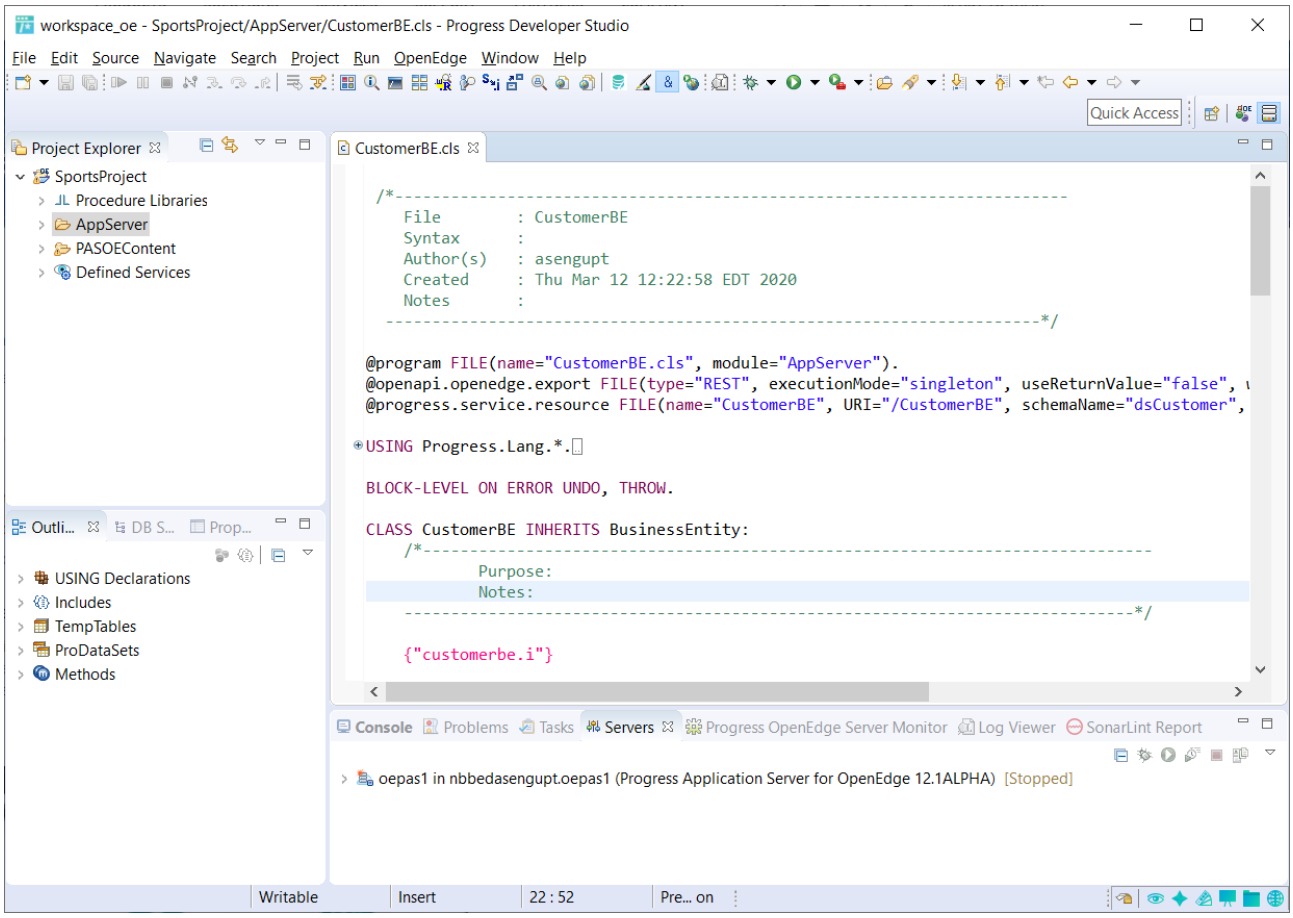
- Set up a development environment
- Decouple ABL business logic from UI code
- Create ABL services
- Connect ABL services to UI

Set up a development environment

Progress Developer Studio for OpenEdge is a modern development environment that integrates with build automation tools, enabling you to develop, test, and deploy changes quickly and efficiently.

Progress Developer Studio for OpenEdge

Progress Developer Studio for OpenEdge is an Eclipse-based integrated development environment that is optimized for creating OpenEdge ABL applications. Developer Studio for OpenEdge comes with a host of features that are aimed at improving developer productivity, such as wizards to create ABL classes, procedures, and services; intelligent code completion for ABL and ABL unit testing; and tools to work with databases and to debug your application code.



Here is an introductory video that provides a tour of the Developer Studio for OpenEdge environment:

Getting Started with Developer Studio

In Progress Developer Studio for OpenEdge, you begin development by creating an OpenEdge project. An OpenEdge project is a container for ABL procedures, classes, services, and configuration files. The OpenEdge project can be packaged as a web application archive (WAR) file and deployed to a PAS for OpenEdge instance as an ABL web application.

After you create an OpenEdge project, you can import ABL code files from your existing application, or create new code files to begin your modernization process.

To try Progress Developer Studio for OpenEdge, you must get the OpenEdge Developers Kit (OEDK), which includes a development instance of PAS for OpenEdge, an OpenEdge database, and other tools for easy deployment and testing.

To learn more about Progress Developer Studio for OpenEdge, see *Progress Developer Studio for OpenEdge Online Help*.

Integration with build automation tools

Progress Developer Studio for OpenEdge integrates with the following build automation tools that enable a CI/CD pipeline:

- **Progress Compile Tool (PCT)**: PCT is a set of Apache Ant tasks created by Riverside Software that automates work in OpenEdge. PCT can be used on its own with ANT, or in a customized build environment using Gradle or another solution.
- **Unified Build and Packaging Framework (UBPF)** for Gradle: The Unified Build and Packaging Framework is a Gradle plugin for OpenEdge applications that supports building projects across platforms and enables smooth transitions between phases of a CI/CD pipeline. UBPF is designed to work with Progress Developer

Studio for OpenEdge and can be used with any project created or imported into Progress Developer Studio. UBPF builds off of PCT to facilitate transitioning between multiple environments.

- **Latte**: Latte is a branch, created by Progress, of the Grabl project that integrates PCT and Gradle. Latte includes ABL support for Gradle, and provides gradle tasks to compile ABL code and run unit tests using ABLUnit.

Decouple ABL business logic from UI code

OpenEdge Application Architecture (OEAA) emphasizes the separation of business logic code from UI code. Doing so enables those parts of your application to evolve independently in response to changing requirements. It also makes your application code easier to maintain.

Review the GUI or character-based user interface (ChUI) components in your application. If they contain business logic or data access logic, extract those parts of the code and put them in ABL classes or procedures. As you do this, you may want to consider using ABL Business Entity classes, which enable you to easily implement Create/Read/Update/Delete (CRUD) operations on your data resources.

After your business logic is isolated in its own layer as a set of ABL classes or procedures, you can create ABL services that connect the business logic with different types of client applications, ranging from your existing GUI and ChUI clients to new web-based clients that you want to develop as part of the modernization project.

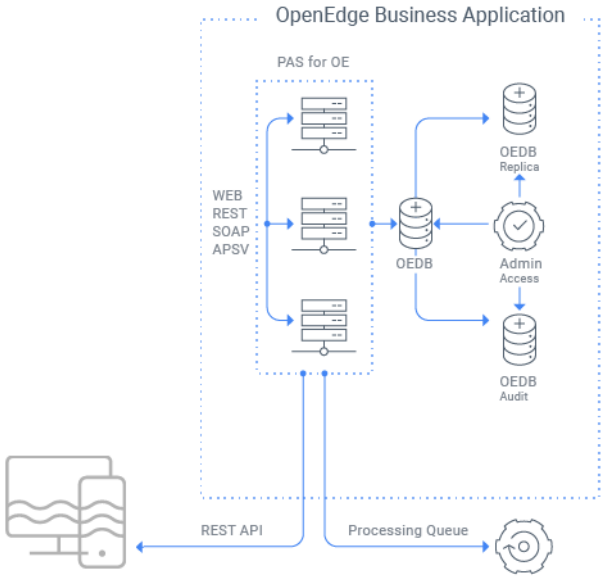
Business Entities

Depending on your use case, an ABL Business Entity may meet your needs. A Business Entity is an ABL class that enables you to easily implement CRUD and Submit operations on data resources. ABL Business Entities can also be exposed as Progress Data Object Services, which are used by a JSDO client application.

To learn more about creating ABL Business Entities, see "Developing Business Entities for Progress OpenEdge Data Object Services" on the Progress Education Community site.

Create ABL services

An ABL service exposes the functionality contained in the business logic layer as APIs for client applications to consume. Each service is bound to one of the following PAS for OpenEdge transports—REST, WEB, SOAP, or APSV. You can create multiple services for the same business logic, each serving a specific type of client application.



Create REST services

REST services are the preferred way to integrate ABL business logic with client applications. REST services are scalable and easier to develop because they use standard HTTP request and response mechanisms.

The REST and WEB transports in PAS for OpenEdge enable you to develop REST services; however, Progress recommends using the WEB transport because it provides more flexibility and supports all HTTP content types.

To create an ABL service based on the WEB transport, you use a web handler as the service interface. A web handler is a custom ABL class that inherits the `OpenEdge.Web.WebHandler` class and contains methods that map to HTTP verbs (`HandleGet()`, `HandlePost()`). When you create the service, you define a list of URIs that get associated with the web handler. At runtime, when a client application makes a request to a URI, the request is passed to the associated web handler and processed by the method that corresponds to the HTTP verb used in the request (`GET`, `PUT`, `POST`, or `DELETE`).

For more information, see the 'Web Handler' in *Develop ABL Services* documentation.

Create Progress Data Object services

Progress Data Object services are specially configured services that use either the WEB or REST transport to enable client access to OpenEdge data and operations. Although Data Object services can be used by a variety of web and mobile client applications, they are optimized for use with JavaScript Data Object (JSDO) clients. The JSDO is a client-side library whose purpose is to provide an easy-to-understand API for querying, creating, updating, and deleting Progress Data Objects and invoking server-side business logic.

For more information about creating ABL services, see 'Develop services' in *Develop ABL Services*. For more information about JSDO, see 'Overview of Progress Data Objects, Services, and Catalogs' in *Develop Progress Data Objects*.

Write ABL procedures to manage request and session context

You may want your application to handle the context in a client request before the request is processed by an ABL service. Context is any data that must persist over several interactions between a client application and the service. This includes things like the user profile, role information, and language preferences. In PAS for OpenEdge, you can manage request and session context by writing event procedures—procedures that are triggered by events such as session startup or shutdown, session connect or disconnect, and agent startup or shutdown.

PAS for OpenEdge provides two application models for client applications to communicate with a PAS for OpenEdge instance: session-free and session-managed. In a session-free model, each request from a client is processed in an independent session by PAS for OpenEdge. Context that must be maintained over several requests must be supplied with each request. Conversely, in a session-managed model, multiple client requests can be handled within the same session.

For more information about managing context, see 'Programming for a PAS for OpenEdge application model' in *Application Development with PAS for OpenEdge*.

Deploy ABL services to PAS for OpenEdge for testing

After you develop ABL services in your OpenEdge project, you can test them either by deploying the services to a PAS for OpenEdge instance in your local OpenEdge development environment or as part of a CI/CD pipeline, where the services are packaged into a web application and deployed to a remote PAS for OpenEdge instance.

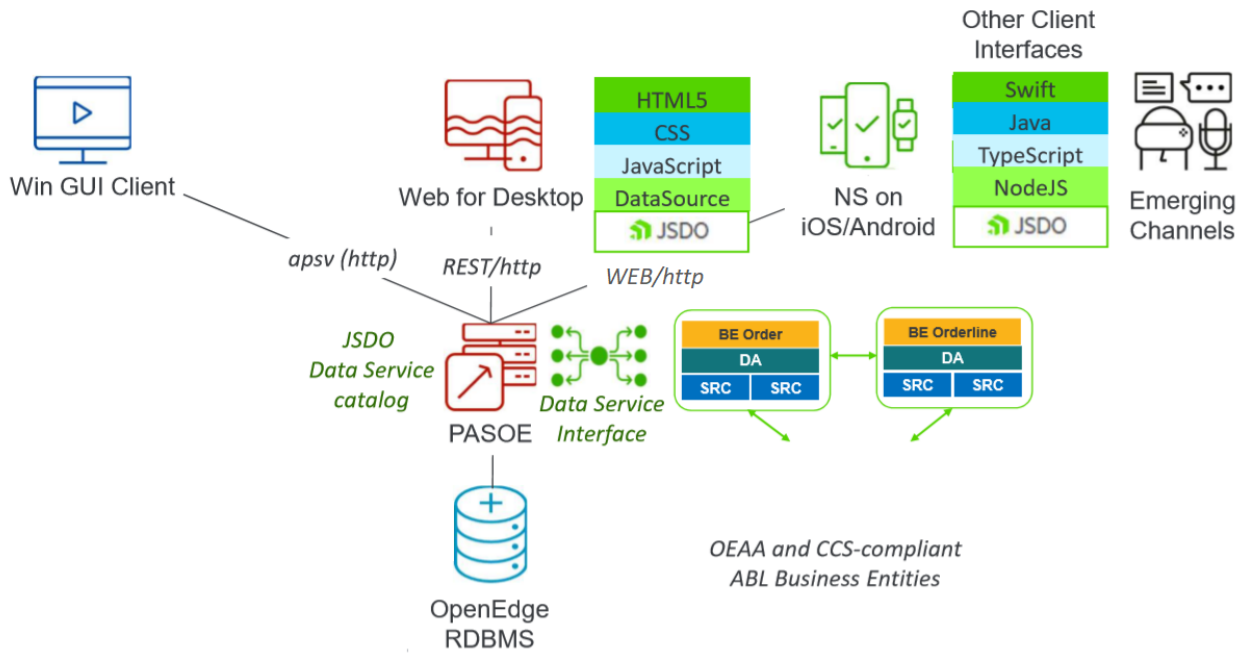
A Progress Developer Studio for OpenEdge installation includes a development version of PAS for OpenEdge that you can use for testing. This video provides an overview of using PAS for OpenEdge in Progress Developer Studio for OpenEdge:

Servers view representation from Progress Developer Studio for OpenEdge

Application migration from OpenEdge AppServer to PAS for OpenEdge

If you migrate your application from OpenEdge AppServer (classic AppServer) or the WebSpeed Transaction Server to PAS for OpenEdge, you must make a few code and property changes. For more information about these changes, see 'Learn About 'Migrating Classic AppServer Applications to PAS for OpenEdge' in *Migrate Classic AppServer Applications to PAS for OpenEdge*.

Connect ABL services to UI



After your ABL services are up and running on PAS for OpenEdge, they can be accessed by a variety of client applications.

ABL client applications, which may include GUI, ChUI, and .NET GUI, access service interface procedures through the APSV transport.

REST client applications, which include mobile and web clients, and are built using a variety of technologies such as Kendo UI, Angular, React, etc, access ABL services through either the WEB or REST transports.

V

Learn more

For details, see the following topics:

- Contact Progress Professional Services for guidance

Contact Progress Professional Services for guidance

The Professional Services team at Progress has experts who can help OpenEdge partners and customers modernize legacy applications. To get assistance with your modernization project, click **CONTACT US** on the OpenEdge Modernization page.