

White paper

# Low-code development

Build applications more quickly and easily  
through code reuse and simplified practices

---

This white paper presents an overview of the low-code development capability in the new OpenText™ AppWorks Application Platform. This low-code development environment enables developers to more quickly and easily build applications through code reuse and simplified development practices.

---

## Contents

Introduction	4
What is low-code?	4
Introduction to AppWorks Platform	4
Workspace	4
Projects	5
Documents	5
Sample projects	7
Information-centric low-code application development	8
Building blocks	8
Basic	9
Process flow	9
Business logic	9
Presentation	10
Content and collaboration	10
Integration with other information repositories	11
Imported entities	11
EIS entities	11
APIs	12
Technology connectors	13
Panels	13
Expression operators and functions	13
Building applications	13
A closer look at creating applications	14
Step 1: Create entities	14
Step 2: Establish the relationship between entities	14
Step 3: Add properties	14
Step 4: Create the user interface	14
Adding functionality to the application	14
Making application-level customizations	15
How a developer uses entity modeling to build a complex application	15
Process-centric low-code application development	15
Orchestration scenario	16
Model-assisted code-centric development	18

## Contents

Content systems integration: Document Store and Extended ECM	18
Standard domain model	20
APIs overview	21
Multi-tenancy and customization	21
Modeling and deploying mobile apps	22
Reporting, analytics and Artificial Intelligence	22
iHub	22
Process Intelligence	22
Magellan	23
Content storage considerations	23
Application deployment	23
Conclusion	23
Appendix A - Data domain models and properties	24
Identity component	24
Domain model overview	24
Domain model diagram	24
Detailed domain model of Person entity	25
Case Management Application Accelerator	25
Domain model overview	25
Help Topic domain model overview	26

## Introduction

Applications vary from simple to complex and natively mobile to mobile-friendly, and in each case, are defined as a self-contained, purpose-specific “program” that helps end users interact with information from multiple systems to drive a business action or outcome. A platform delivers the tools, capabilities and features a developer needs to build an application, regardless of its complexity. This paper discusses the various development approaches AppWorks supports and the tools and capabilities it provides. In addition, it provides examples of how developers can leverage low-code to build applications.

## What is low-code?

Low-code is an emerging development approach that reduces the need for a developer to write new code every time they build an application. Instead, low-code development environments provide reusable, pre-built components with “drag and drop” capabilities that minimize the coding effort and speed time to deployment. Using this simplified approach, developers can more easily collaborate with subject matter experts throughout the development process and business users can work on the application themselves, reducing development cycles and the pull on IT resources.

Low-code can be used to build highly complex, enterprise-caliber applications. The OpenText approach to low-code supports many development approaches including: information-centric low-code, process-centric low-code, model-assisted coding and traditional development. This empowers the professional developer to build the optimal application(s) for their business. The focus of this white paper is on the three low-code approaches:

1. **Information-centric low-code:** Start by modeling the data and the data that resides in the business, and then determine the lifecycle of the data and how this data flows through the business
2. **Process-centric low-code:** Begin with the business process model and model the processes executed in the business domain and then determine which systems, services and actors are involved in the various processes
3. **Model-assisted coding:** Model applications using a separate IDE and then add generated artifacts to the project along with other AppWorks models

## Introduction to the AppWorks Platform

AppWorks provides a set of integrated components that help deliver seamless, reliable and high-performance design and runtime environments for conducting enterprise business.

This section describes many of the platform components that are accessible to the low-code developer. These components are presented in the order that a developer would engage with them to build an application for the first time.

### Workspace

A workspace is the starting point for the low-code developer. A workspace contains all the **projects** with which a developer works.

In most organizations, there isn't just one developer working in a workspace, but rather, there is a development team. The platform supports various ways to configure workspaces to support development teams. The best practice is to create a separate workspace for each developer on the team, all connected to a common SVN source control repository.



When used this way, each developer has a separate workspace isolating their changes. SVN allows standard version control so that developers can retrieve changes from others and check in their changes to make them available to other developers.

Developers can also leverage applications from others in the Imported Application Packages area. Developers do not have the source code for these packaged applications. These may be purchased applications or applications from other teams in the same enterprise. An example of an OpenText application that is available out of the box in the Application Packages area is the Identity Package (see [Standard domain model](#)).

## Projects

A workspace contains any number of projects. A project corresponds roughly to an application. More specifically, a project is the unit of packaging and deploying<sup>1</sup>—a developer can package a project that produces a file that can then be uploaded to a system and deployed. This packaging concept supports best practices associated with governance of applications<sup>2</sup> (DTAP).

Projects can vary from simple to complex, depending on the type of application. The platform provides options to help the developer organize their projects to make their applications easy to maintain. The following example shows an application that has several projects.



## Documents

A project contains a set of artifacts called **documents**. Many kinds of documents can be added to a project and the list in the table below will continue to grow as new capabilities are added to the platform. There are more than 60 document types, however many of them are not frequently used and will be deprecated over time. The following table summarizes the most frequently used and valuable document types.

<sup>1</sup> Projects can also be referred to as a package. A package typically represents an application that has been deployed.

<sup>2</sup> Controlling who can deploy to production, ensuring that the package deployed to production is the same as the package that was tested, etc.

<b>Entity modeling</b>	Document types used in information-centric, low-code applications
<b>Entity</b>	Defines an entity. An entity represents a business object that is used by an organization. <a href="#">See Information-centric low-code application development.</a>
<b>Home page layout</b>	Used to provide users with a landing page
<b>Theme</b>	Defines a theme specifying the overall “look” of the end user interface
<b>History log</b>	Defines a log file that tracks specific changes or access to items to maintain a collection of history events
<b>Process</b>	Document types used in process-centric, low-code applications
<b>Business process model</b>	Defines a business process model
<b>Business identifier</b>	Defines and uses business identifier
<b>Decision table</b>	Defines a set of policies
<b>Schedule</b>	Defines repeating business events
<b>Developer</b>	Document types used in model-assisted and traditional applications
<b>Cascading style sheet</b>	Adds a CSS file to a project
<b>JavaScript Message Bundle</b>	Adds a JavaScript message bundle that can be localized to a project
<b>JavaScript source</b>	Adds a JavaScript source code file to a project
<b>HTML source</b>	Adds an HTML file to a project
<b>Java source</b>	Adds a Java source file to a project
<b>Java archive</b>	Adds a Java archive (JAR) file to a project
<b>Java class metadata</b>	Generates web services on a Java class
<b>Web library definition<sup>3</sup></b>	Adds a folder to manage web resources to a project
<b>XML</b>	Adds an XML file to a project
<b>XML schema</b>	Adds an XML schema file to a project
<b>Developer</b>	Document types used to integrate with other systems <sup>4</sup>
<b>EIS<sup>5</sup> connector</b>	Enables developers to create a connector (by writing Java code) to use data of an external system as entities
<b>EIS<sup>6</sup> repository reference</b>	Defines a connection to a specific repository using an EIS connector
<b>Specific EIS connectors</b>	Provides packaged EIS connectors for OpenText™ MBPM and OpenText™ Case360
<b>Application connector</b>	Enables developers to create a connector (by writing Java code) to interact with an external system through SOAP web services
<b>Web service</b>	Creates a web service to access business logic from various sources such as Database, WS-AppServer Package, external WSDL and Java classes
<b>WS-AppServer Package</b>	Enables a developer to create web services by writing Java code. Commonly used to develop web services on top of relational databases.
<b>Database metadata</b>	Represents an imported database schema that can be used to import entities or generate a WS-AppServer package
<b>Developer</b>	Document types used to integrate with other systems <sup>4</sup>
<b>Business calendar</b>	Defines a business calendar used to perform business day calculations
<b>Role</b>	Defines a role to be used to control access to resources

<sup>3</sup> This is also used with entity modeling to manage image resources.

<sup>4</sup> There is some overlap with entity modeling, as the EIS document types also provide integration features.

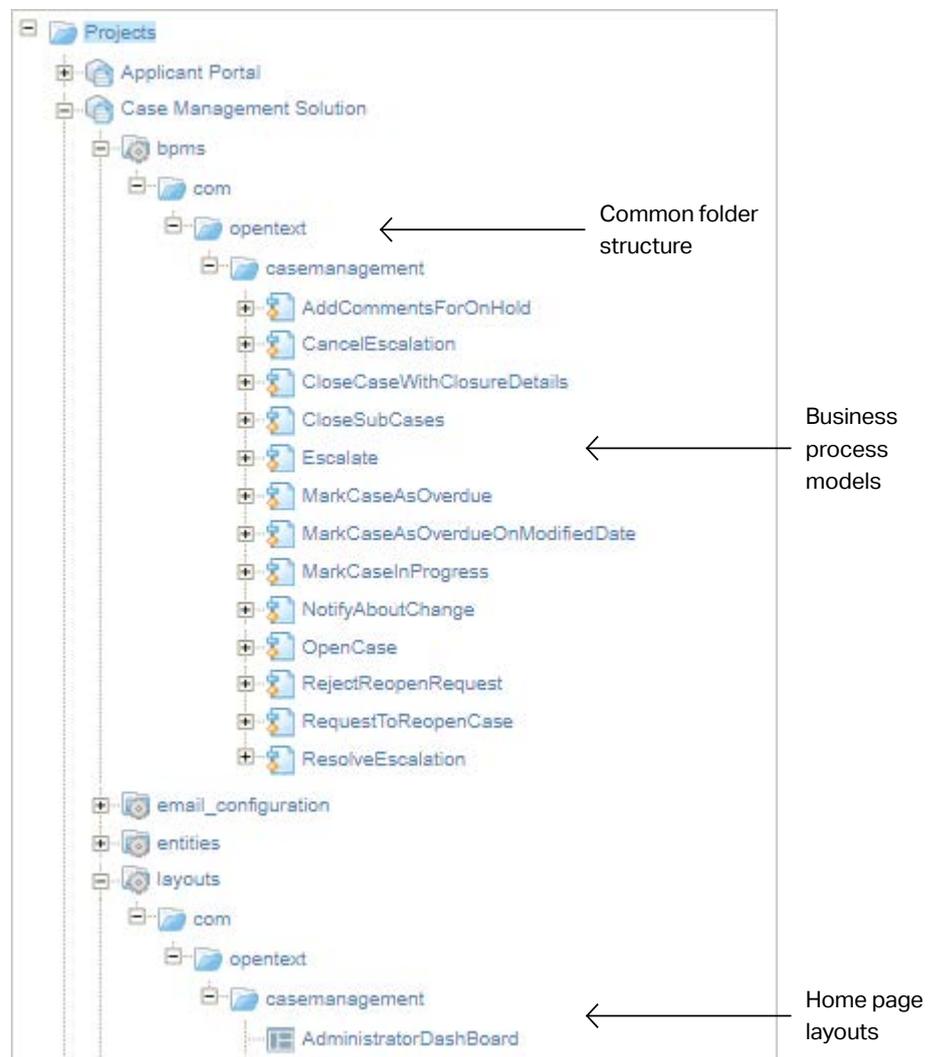
<sup>5</sup> Enterprise Information Service (EIS) entity connectors provide access to information in other systems.

<sup>6</sup> Enterprise Information Service (EIS) entity connectors provide access to information in other systems.

In practice, while many projects are applications, some projects exist solely to define artifacts (documents) that are shared and leveraged by other applications. For example, the Identity Package is a project that is not an application in isolation, but provides artifacts used by other projects.

### Sample projects

As mentioned above, projects can vary from simple to complex, depending on the type of application. For example, a simple application might have only one or two projects, whereas a more complex application might include numerous projects. The following example shows an expanded view of a low-code application structure. To help make complex projects easier to maintain, developers create a common folder structure inside each project and establish naming conventions in the beginning of the development cycle.



## Information-centric, low-code application development

AppWorks supports information-centric design by breathing new life into the proven methodology of object-oriented programming with entity modeling. This approach is used in cases where there is a clear view on how the business data is structured. With information-centric design, developers first model the objects or entities that reside in the business domain, then determine how they relate to each other and then specify the various pieces of business logic for these entities and design how end users will interact with them.

An entity represents a business object that is used by an organization. For example, entities for a case management application might be Case, Requestor or Priority. Entities are typically related to each other, as when a Case is related to the Requestor (person) who has made the request.

Entity modeling offers a compositional approach to application development. Instead of programming certain pieces of functionality repeatedly, a developer defines the capabilities of an entity by adding functional modules (called building blocks) to it and then configures them to suit their needs.

Entity modeling offers low-code and compositional development capabilities that simplify application development in various ways:

1. Introduces an intuitive way of modeling the business domain that is close to how the subject matter expert conceptually thinks about it
2. Guides the subject matter expert when navigating through the domain model to express business logic
3. Provides a simple way to add behavior to an application by enabling a subject matter expert to select and configure functional modules (called building blocks)
4. Enables an application to access data that resides in an external system

With this approach, a developer can start by adding entities to their project. An entity is one of the types of "documents" that can be added to a project. See [Documents](#).

### Building blocks

With an understanding of entities, a developer can start to think about other elements to build onto their entities. These elements are called building blocks.

A building block adds pre-built functionality to an entity. Out of the box, AppWorks provides a library of building blocks. For example, adding the Discussion building block to an entity adds a threaded discussion form to it.

The following table describes the current set of building blocks. This list grows with each release, as OpenText adds functionality to the platform.

Building blocks are categorized into the following areas:

- Basic
- Process flow
- Business logic
- Presentation
- Content and collaboration



The order in which a developer adds building blocks to entities is not pre-defined. In addition, some building blocks, like History, can only be added to an entity once, while others can be added any number of times, such as Properties and Forms.

### Basic

Basic building blocks provide the foundation for an entity.

Building block	Description
<b>Property</b>	Adds a property to an entity. Properties can have various types (text, float, enumeration, currency) and can specify constraints such as minimum value or pattern
<b>Relationship</b>	Specifies a relationship from an entity to another entity. Relationships can be one-to-one and one-to-many and many-to-many and peer-to-peer, as well as parent-child
<b>History</b>	Tracks changes to an instance of the entity in a history log (an audit trail)
<b>Security</b>	Offers the ability to define a security policy that controls access to instances of an entity and various building blocks specify permissions that can be granted to roles in the security policy
<b>Identity</b>	Adds properties that identify entity instances of the entity—all entities include this building block

### Process flow

Process flow building blocks define information about business processes.

Building block	Description
<b>Lifecycle</b>	Adds a lifecycle in the form of a set of linked states and activities to an entity, making the entity into a case
<b>Assignee</b>	Provides the ability to assign instances of the entity to individual participants or roles, indicating some form of responsibility for that item
<b>Activity flow</b>	Enables a designer to define simple sequences of tasks that can be added to a case on an ad-hoc basis
<b>Tracking</b>	Includes date/time created and last-modified and user who created/last modified an instance of the entity

### Business logic

Business logic building blocks define business logic and integration with other parts of the platform.

Building block	Description
<b>Rule</b>	Adds a rule to an entity to specify business logic or a constraint or to create a custom action
<b>Web services</b>	Integrates an application with other systems or parts of the platform using web services
<b>Deadline</b>	Tracks completion of various activities and objectives of an entity instance against a deadline

### Presentation

Presentation building blocks define the user interface for the application.

Building block	Description
<b>Layout</b>	Specifies an overall presentation of an instance of the entity. Multiple layouts may be defined for use by different personas or different states of an instance of the entity.
<b>Form</b>	Specifies a presentation for the entity's (or related entity's) properties. Multiple forms may be defined for use in different layout panels.
<b>List</b>	Allows participants to access lists of instances of the entity. Any number of lists may be added to an entity.
<b>Title</b>	Adds a title that can be used to identify instances of the entity. Having a title on an entity enhances the readability of history logs, etc.
<b>Action bar</b>	Displays customized action buttons to users.
<b>Mobile app</b>	Creates a mobile app based on the application. <a href="#">See Modeling and deploying mobile apps.</a>

### Content and collaboration

Content and collaboration building blocks provide users with features to manage content and communicate about items.

Building block	Description
<b>File</b>	Enables users to add a single file to an item and, with OpenText™ Content Server, users have additional options, such as checking in and checking out versions of an item
<b>Content</b>	Offers the ability to associate any number of files with an entity instance
<b>Business workspace</b>	Adds OpenText™ Extended ECM (xECM) functionality to the entity
<b>Discussion</b>	Enables participants to conduct discussion associated with an entity instance
<b>Email</b>	Allows the use of email in an entity, including both sending and receiving emails from an entity instance
<b>Email template</b>	Includes an email template with an entity that can be used to send emails with embedded entity data

The following examples show how building blocks can extend the entity:

- Lifecycle: Includes many properties, including CurrentState and last performed task
- Relationships: Adds the CreatedBy and ModifiedBy relationships to User with tracking
- Behaviors: Ensures the associated file is deleted when an instance is deleted
- Actions: Adds a View History action
- Permissions: Provides permissions controlling who can upload or download the file
- Panels (UX): Adds Discussion to the Discussion Panel
- APIs: Adds the ContentStream REST Resource to upload/download file content

## Integration with other information repositories

Applications are not created in a vacuum. In fact, an application is built to enable the user to act on several types of information to address a specific issue (claims approval, for example). Thus, applications must be designed to work well with existing information repositories. AppWorks provides several standards-based or pre-built approaches for building applications that leverage information from multiple repositories.

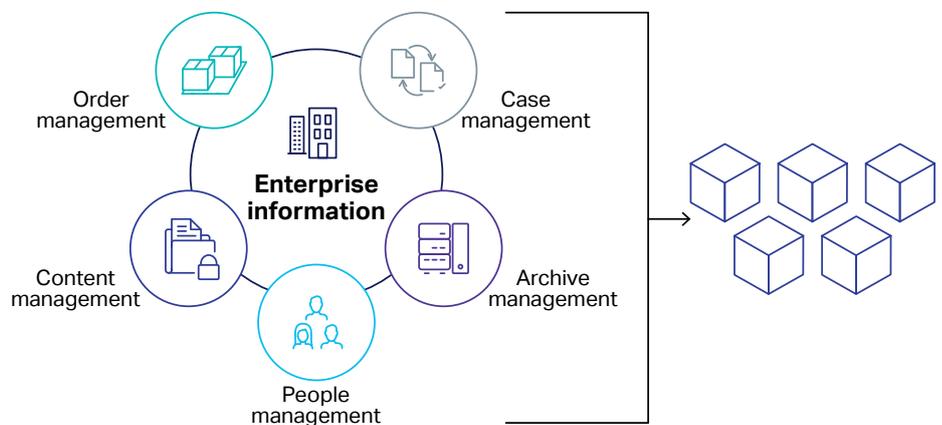
### Imported entities

If the information needed by the application is accessible directly as a database table, a developer can simply import the database table(s) into their project as entities. These entities can then be used much like the entities created in the project<sup>7</sup>. If multiple tables are imported, AppWorks will automatically define relationships between the tables from the foreign key constraints in the database.

### EIS entities

If the information needed by the application is only available through APIs, the developer may choose to create an EIS connector for that repository. An EIS connector provides the information from another repository to the project by adding appropriate entities to the project. These EIS entities can be used much like the entities created in the project<sup>8</sup>.

The target external systems for a custom EIS connector include enterprise applications (ERPs such as SAP<sup>®</sup>) and repositories (for example, SAP<sup>®</sup> SuccessFactors<sup>®</sup> or Microsoft<sup>®</sup> SharePoint<sup>®</sup>). As each of these systems holds information, that information can be used in an application. An application developer can focus on building their application without having to create the connection to an information repository.

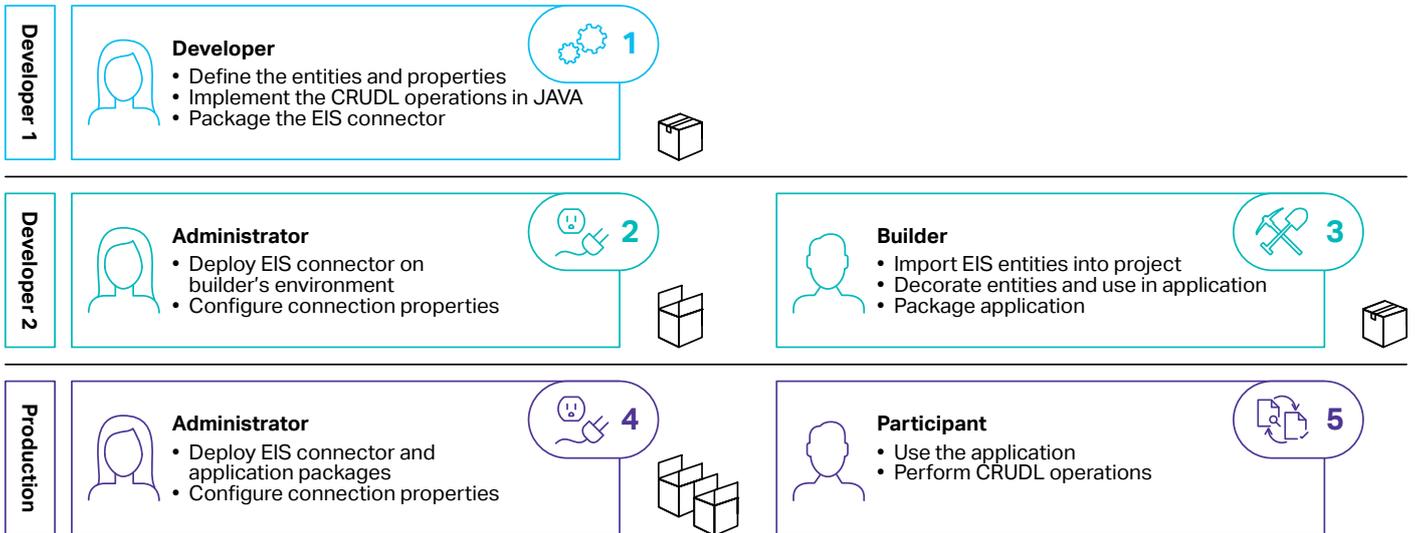


The structure of an EIS entity is determined by the target repository and it cannot be modified. That is, a developer cannot add, remove or modify its structured building blocks. But they can add and modify presentation building blocks and build the user interface just as they would for a native entity. Once the presentation is added, they can access an EIS entity via OpenText™ Process Experience, just as they would access native entities.

<sup>7</sup> The primary difference is that imported entities cannot be modified as OpenText does not own the table definition.

<sup>8</sup> The primary difference is that EIS entities cannot be modified as they must conform to the information model in the external repository.

Creating an EIS connector requires coding, but once the connector has been created, a low-code developer can use them quite easily in their application. As explained in the below diagram, EIS connectors developed for a particular system can be reused for different instances of that system and can be reused by multiple applications that might use different entities from this external repository.



AppWorks provides a growing set of EIS connectors out of the box to support many developer scenarios exposed via APIs, building blocks and web services.

To help customers move from other OpenText digital process automation (DPA) products, EIS connectors have added capabilities. Out of the box, AppWorks provides connectors to:

- OpenText MBPM
- OpenText Case360

**APIs**

If the external system exposes an API that cannot be easily wrapped with an EIS connector, the developer can use a business process model to retrieve information from the external system or orchestrate transactions against it.<sup>9</sup>

API	Description
<b>SOAP</b>	AppWorks provides a set of SOAP APIs that a developer can expose for use by coders creating applications that leverage low-code
<b>Web</b>	AppWorks delivers a set of APIs that a developer can use in their application to define its interaction patterns with the backend API, enabling an application to communicate with the backend for processing information.
<b>REST</b>	AppWorks includes a set of APIs that offers access to instances, task management and Case activities/details
<b>Entity REST</b>	AppWorks has an API that consists of URIs that identify resources on the server where a developer can use the standard HTTP methods (the uniform interface).

<sup>9</sup> All available APIs will be documented in a future release.



## Technology connectors

AppWorks supports the following technology connectors that can be used to interact with external systems:

- Database connector
- Document Store connector
- Email connector
- FTP connector
- Java connector
- SOAP connector
- REST connector

## Panels

With this extensibility point, a developer can create panels to use in a layout. There are two types of panels, entity instance and home page. Entity instance panels can only be used on layouts on an entity. Home page panels may be used on both home page layouts (dashboards) and entity layouts.

An entity instance panel can be configured so that it is only available in the layout designer if a specific building block is present in the entity. As an example, the Tasks panel is only available if the Lifecycle building block is present in the entity.

## Expression operators and functions

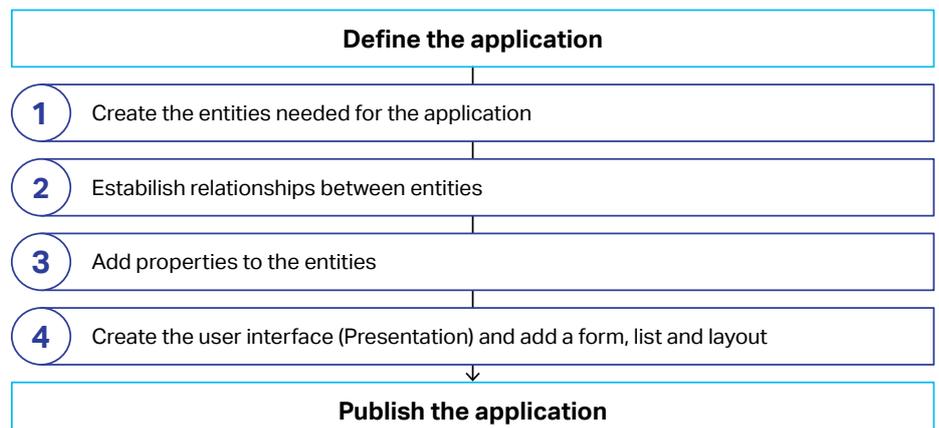
Using this extensibility point, a developer can add operators and functions to the entity expression language. Functions can have various implementations, including rule expressions, web services and JavaScript.

## Building applications

Before a developer begins building an application, they need to think about what they want to build. It is important to think about what problems it needs to solve, what information they need to collect and act on, what business processes they can automate, how users will use it and what they need to see. Thinking in this way, will help a developer build a solid information-based application domain model.

The next step is to start building it.

A low-code developer can create an application in four steps.



## A closer look at creating applications

This section takes a closer look at the steps involved to create an application. Once an application takes shape, a developer can add more functionality based on their requirements.

### Step 1: Create entities

A developer begins creating an application by building entities. They must think of the entities as the information they want to capture in their application. For example, an automobile claims application might have Claim, Policy, Vehicle and Policy Holder entities. A healthcare application might have Physician, Patient, Allergy and Medication entities. An HR staffing application might have Job and Candidate entities.

### Step 2: Establish the relationship between entities

Not all entities need to be related, but most of the time when building an application, one or more of the entities are related in some way. So, after creating the entities they want in their application, the developer must think about how they are related. For example, in an automobile claims application, when a claim is created, the user needs to be able to select the policy holder to create the claim. Therefore, the Claim entity must have a relationship to Policy Holder.

### Step 3: Add properties

After entities are defined, a developer can add properties to the entity. Properties in a Claim entity in the automobile claims application might include, Date of Accident, Accident Location and Driver.

### Step 4: Create the user interface

Creating the user interface involves three steps that bring the application to life for users. Typically, to begin, at least one of the following is added:

1. A form: Drag and drop the properties to design the form that users will use in Process Experience
2. A layout: Design the layout with the panels for users to see. Often layouts display a form, an action toolbar, breadcrumb panel and more (based upon the building blocks used in the entity).
3. A list: Add the properties to display to users in the list. A list displays the existing entity instances. For example, an All Jobs list in an HR staffing application would include a list of all the job openings. This list might include properties, such as Job Title, Summary, Date Opened and Hiring Manager.

## Adding functionality to the application

With just four steps, an application begins to take shape. A developer can then think about other functionality to add to the application. Options include:

- Business logic using rules
- The ability for users to attach files via the Content panel
- A workflow using the Lifecycle or Activity Flow
- The ability to show an audit trail using History
- Security permissions
- Collaboration capabilities using Discussion

At this point, a developer can leverage the pre-built functionality that comes in the building blocks to extend the application to meet their needs. Most often, they won't use all building blocks in all entities. In a typical application, primary entities will include more functional building blocks than supporting entities.

### **Making application-level customizations**

All the above steps describe the functionality added to entities. However, functionality can also be created and customized at the application level. For example, a developer might apply their company's color scheme with a custom theme, add their company logo to the header or create a customized home (landing) page for users.

In addition to the customizations mentioned above, a developer can add a structured, long-lived or short-lived business process to the application. These processes can interact with web services and entities, allowing a developer to take advantage of any outside services needed to accomplish their application requirements.

### **How a developer uses entity modeling to build a complex application**

Applications can quickly become complex by the sheer number of items added to the entities. Consider a simple application that has a few entities, five relationships to other entities and only a handful of business rules. In a complex application, a single entity might have 26 relationships and 30 to 35 business rules. In addition, complex applications often include various structured, long-lived and short-lived business processes, integration with external systems using web services and external user interfaces.

It is important for a developer to consider best practices when developing applications as they often start out simple and then quickly grow complex. To help make the development and application maintenance easier over time, a developer should define naming conventions and project organization best practices before developing an application.

### **Process-centric low-code application development**

With AppWorks, process-centric applications are created by adding one or more business process model documents to a project. Developers edit these documents and build a process model using a modeler that uses industry standard Business Process Model and Notation (BPMN). AppWorks supports the industry-standard XPD L interchange format to import and export BPMN processes from other systems.

Process-centric applications can be used to build applications that include the following use cases:

- Orchestration is used to create services that, when called, perform a series of operations against one or more other services. In this case, the process has few, if any, user activities.
- Structured human interaction is used when a business process is fully understood<sup>10</sup> and the developer wishes to completely control the way work is performed.

Most process-centric applications include aspects of both orchestration and structured human interaction. Orchestration processes with no user interaction can be exposed as web services, which can be called from external systems or used in other models. Processes can use application connectors to interact with external systems. Out of the box, AppWorks delivers connectors for HTTP (REST), FTP and JMS. There is also a collection of community connectors that includes connectors for SAP and other enterprise applications. Alternatively, developers can create a custom connector through Java code, enabling interaction with any system exposing APIs.

<sup>10</sup>One of the main pitfalls of this type of application is thinking the process is fully understood, when it is not. This is the primary driver for more case-oriented application development where the participants have more control to deal with edge conditions. In today's world, applications with significant human interaction are usually implemented as case-centric applications.

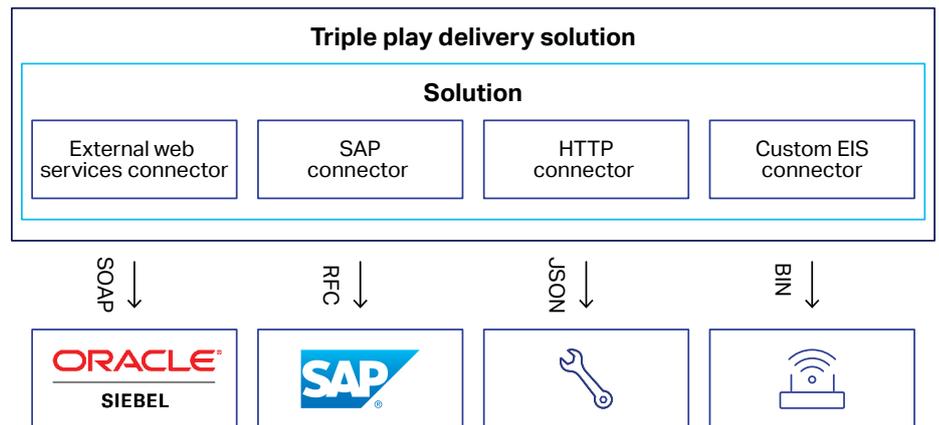
When users interact with process-centric applications, they usually do so via Process Experience using the standard Inbox application included with the platform. In some cases, an application-specific user interface is coded to provide a fully tailored user experience. In other cases, there is no user interaction at all.

**Orchestration scenario**

Because businesses run diverse sets of information systems, often gathered through various acquisitions, new applications must integrate with multiple existing systems. AppWorks is designed to operate in this context. It provides connectivity for various existing systems and a framework to build connectors for yet unsupported systems and includes functionality to develop new applications.

Consider this following scenario: A company builds a solution to provide triple play services to their customers. The solution needs to automate the entire chain from taking the order on a website to activating the internet connection at home. Most of the functionality is available in existing systems and these systems need to be integrated: Siebel® for CRM, SAP for billing, a home-grown system for service engineer planning and another proprietary system to handle the dispatching of physical goods, such as routers.

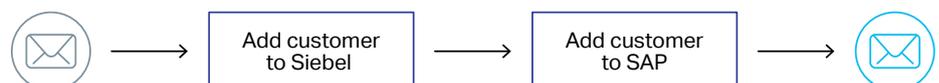
The following diagram depicts the environment.



The solution comprises the following:

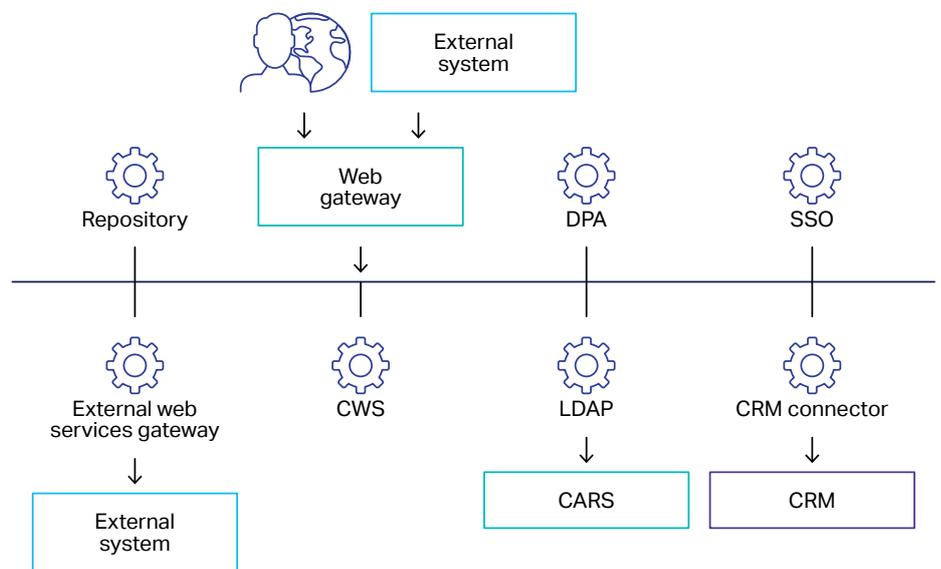
- The order entry form is implemented in ASP.NET, as the company has chosen to implement the customer portal in ASP.NET.
- A set of related entities are created, including the Order entity, which is generated when the customer submits the web form. The Order entity has a lifecycle that describes the state and activities of the order.
- Some composite web services are implemented as short-lived processes to simplify the interaction with Siebel and SAP from the Order lifecycle.
- An EIS connector is built to interact with the service to manage physical goods.
- User interfaces have been created for employees through Process Experience.

Given that every new customer needs to be registered in both Siebel and SAP, the developers built a composite service on top of both products, implemented as a short-lived business process, on top of both systems. The composite service is a regular SOAP web service that is implemented through the model below.

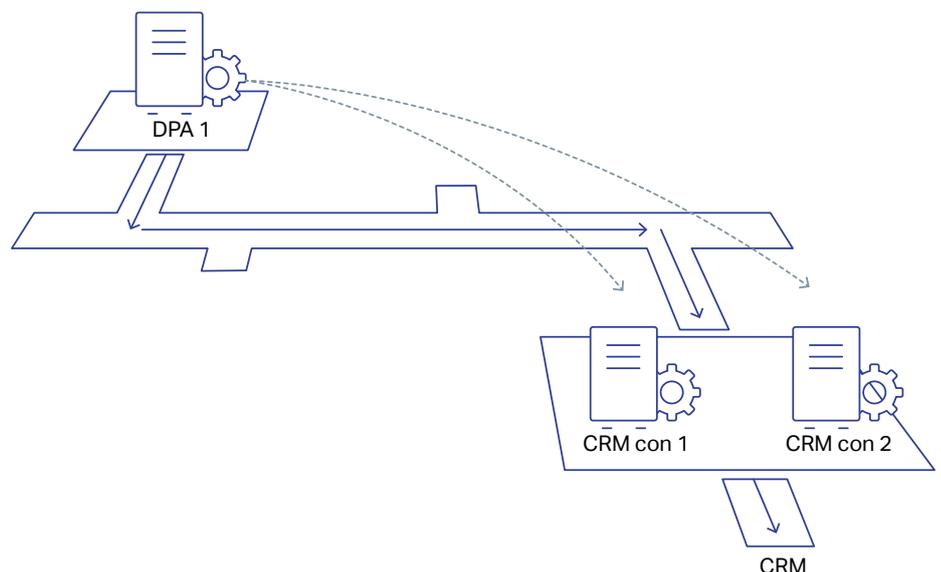


Since Siebel has a good web services interface, it is used for the integration. The web service APIs are imported into AppWorks and invoked through the external web services gateway (often called the UDDI connector). For SAP, the SAP connector is used. The required BAPIs are imported into the platform and invoked from the business processes. The service engineers planning system exposes a set of RESTful APIs, so the HTTP connector is used to expose SOAP web services for these, which are again consumed in the business processes. The physical goods service is totally proprietary and only exposes a Java API. The solution developers implemented an EIS connector, which exposes the physical goods as entities, so they can be related to the rest of the entity model.

The basic communication architecture of the platform resembles a bus, as depicted in the following diagram.



All participants linked to the bus can exchange SOAP messages. Logically, there is a central bus that links all participants, but technically, the communication is direct. The bus framework knows the containers that provide the logical service, which of these containers are available right now and whether these containers are properly functioning. Within that set, the requests are load balanced and directly sent from sender to receiver.



## Model-assisted code-centric development

AppWorks also supports model-assisted code-centric application development. In this mode, most of the application is developed using a separate IDE and the artifacts generated by the developer are added to a project along with other AppWorks models. For example, a developer may use entity modeling to define a data domain that results in a set of database tables, but prefers to implement the business logic using Java code. All current AppWorks Data Domain models are contained in [Appendix A: Data Domain Models and Properties](#).

Each of the different approaches to application development may be combined to create an application. For example, OpenText customer, Monster Energy created a fully custom web experience for their partners, accessing entities defined in AppWorks via their APIs. This is more of a hybrid approach, as the user interfaces for internal users of this application are implemented using information-centric low-code tools.

## Content systems integration: Document Store and Extended ECM

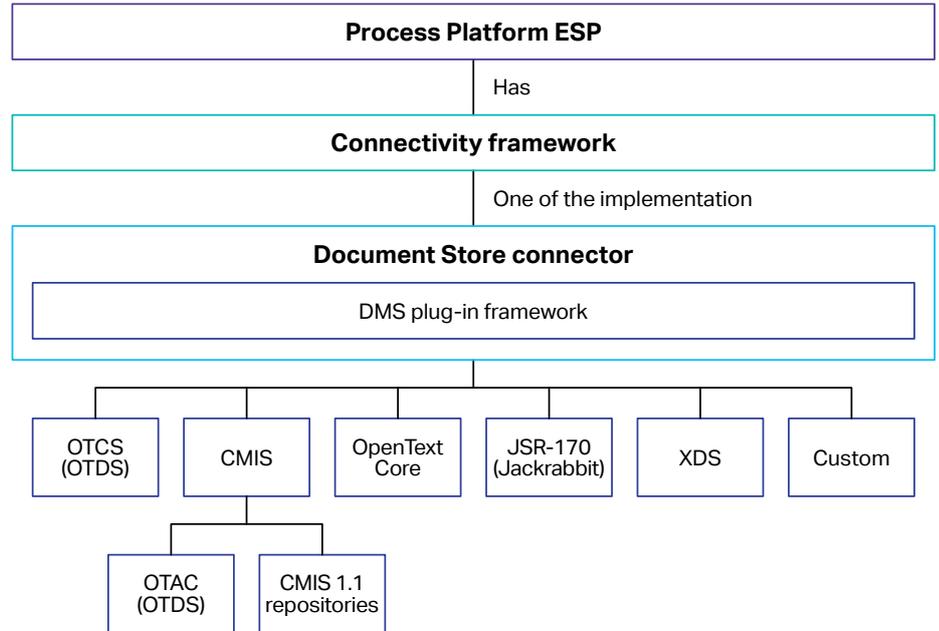
No matter which approach to application development a developer uses, it is critical to integrate with content systems to manage the document content associated with applications. To support this, AppWorks includes the Document Store service, which provides a programmatic abstraction for generic document storage services. With AppWorks, the configuration of the Document Store can be completed post-installation, allowing the developer to focus on building a feature-rich application without worrying about which document management system will be used to run the application. System administrators can configure which document management system to use at runtime.

This abstract interface has multiple implementations, providing seamless integration to Content Server, OpenText™ Core, OpenText™ Archive Server and OpenText™ Documentum™, as well as an implementation on the standard CMIS interface that supports any compliant CMIS 1.1 repository.

This underlying infrastructure is brought in with the entity/case low-code application development tools via the File and Content building blocks that internally use the Document Store connector. With the File building block, a developer can associate a single document with an instance of an entity. With the Content building block, they can associate a collection of documents with an instance of an entity.



Independent of Document Store, which provides generic integration to various content repositories, the entity framework also provides Extended ECM integration using the Business Workspace building block. This building block pairs an instance of the entity with a Content Server Business Workspace using the Extended ECM infrastructure. This provides an exceptionally seamless content experience when it is used in an application.



The Document Management capabilities are exposed to the developer in the following ways:

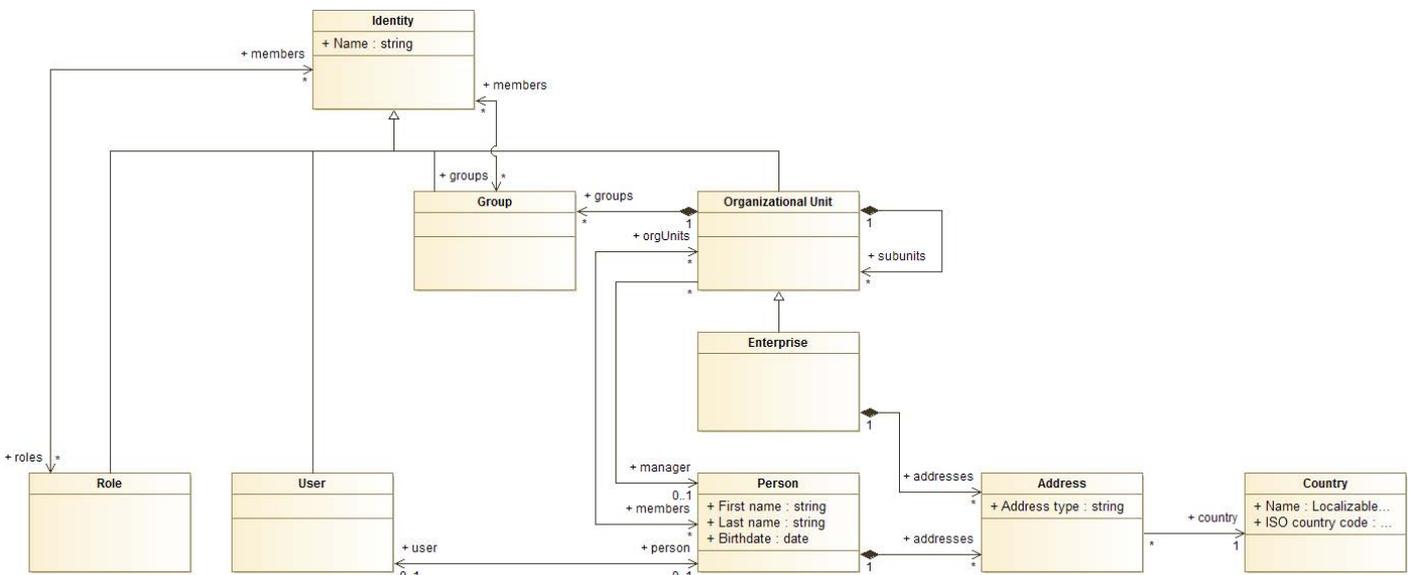
- Out-of-the box support for
  - Content Server
  - Archive Center
  - Documentum
  - OpenText Core
  - CMIS 1.1 compliant repositories
  - Jackrabbit and JSR 170 compliant repositories
- Extended ECM for AppWorks
  - Creates workspace automatically when application objects are made
  - Synchronizes application data with workspace based on events
  - Manages workspace inside application through widgets
  - Establishes relation between workspaces based on relation defined in application
  - Supports business attachments
  - Links with existing workspace in Content Server
  - Shares same workspace created by leading application

### Standard domain model

Virtually every application uses concepts like Person and Enterprise. To prevent developers from having to repeatedly define these and to prevent users from needing to register people and business partners in every application, AppWorks comes with a set of predefined entities, in the form of the Identity Package. The most important entities are:

- **Person:** Represents a natural person. This entity can be used to represent a user, employee, etc.
- **Address:** Persons and enterprises have an address, but the same entity can be used for addresses of warehouses and stops for deliveries.
- **Enterprise:** Represents an organization. This entity is a subtype of Organizational unit, thus inheriting the possibility to have people as members and to define a hierarchical structure. It can be used to represent customers, suppliers, etc.
- **Organizational unit:** A unit of an organization. It defines a hierarchical structure and people can be members of an organizational unit through assignments to positions in the unit.
- **User:** A user of the system. This entity is related to Person, basically giving that person the role of user in the system.
- **Group:** This is a notion from the security system. Commonly, roles are synchronized from an external identity management system, for example, Microsoft® Active Directory®, through OpenText™ Directory Services.
- **Role:** This is a security notion too. Roles are defined by applications and systems to grant permissions to user personas.

The following diagram provides a high-level overview.



Besides these key entities, there are entities for phone number, email address, country and state.

The entities in the Identity Package are provided for reuse by application developers. This out-of-the-box domain model helps speed application modeling and, if used properly, also reduces the administrative burden for users.

Explanations of all current data domain models and their properties are contained in [Appendix A: Data domain models and properties](#).

## APIs overview

Within AppWorks, APIs are essential. Nearly every artifact in a project consumes or provides services. A good example is a business process model, most processes are themselves exposed as APIs that can be called and, of course, a business process model can call other services.

Even a non-coding developer can produce APIs that can be consumed by others. Adding APIs to an entity is a trivial operation that requires virtually no coding skills.

Every process and entity in an application extends the set of APIs available for creating the next application.

## Multi-tenancy and customization

AppWorks is a multi-tenant platform. Applications created through AppWorks can be deployed for all tenants of the system or for a specific tenant. Even if the application is deployed for all tenants, the data of the different tenants are strictly isolated from each other. Tenants can have their own configurations and customizations of applications, allowing them to tailor the system exactly to their needs. The multi-tenancy functions also support deployment in a cloud scenario where some, but not all customers (or end users) may need an application. For example, if you have four customers, an application might be only installed for customer one, two and four, but not three.

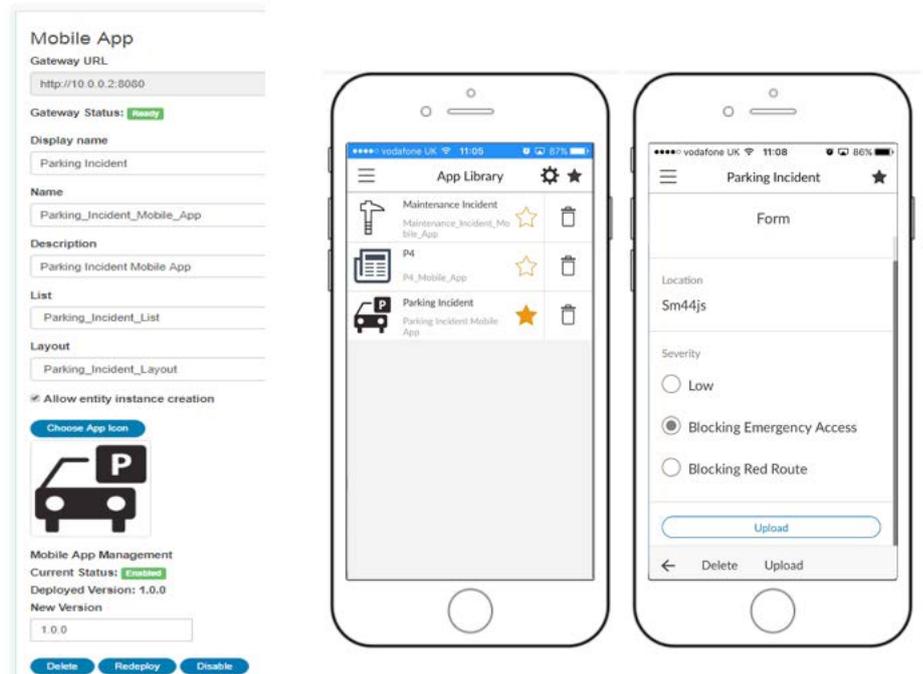
With most applications on the market, customizations could block the road to upgrades, as they modify the original application, making it impossible to accept newer versions from the vendor. But in AppWorks, the customizations are kept separate from the application. The customization is applied to the original application when running it. When the vendor delivers a new version, the customizations can be applied on top of that.



## Modeling and deploying mobile apps

AppWorks allows the mobile 'micro' apps to be modelled in the low-code design environment and then deployed onto users' devices. These apps expose elements of the model: layouts, forms, lists and individual entities and allow the end user to remotely participate in the process by creating new entity instances or performing actions on existing entity instances to change their state.

The Mobile App building block is added to the entity's model and configured as shown below. The app can then be deployed and enabled automatically on the OpenText™ AppWorks Gateway. It then becomes available to mobile users through the AppWorks mobile app.



## Reporting, analytics and Artificial Intelligence

### iHub

AppWorks provides a two-way integration with OpenText™ Information Hub (iHub).

First, for an application built in AppWorks, one can download a Data Object Design file, which can be imported into OpenText™ Analytics Designer. This file contains the connection details, data sources and relationships of the entities in the format used by Analytics Designer. As a result, a report developer does not have to perform the data provisioning, which is done automatically.

Second, AppWorks provides an iHub panel, which allows displaying an iHub report or dashboard on a layout. If required, the iHub panel interacts with its context, as reports can be filtered dynamically based on AppWorks data being displayed.

### Process Intelligence

OpenText™ Process Intelligence provides operational status information and business insights for digital business automation. As Process Intelligence has been built specifically to support process-centric applications, its reports, dashboards and data exploration capabilities provide complete visibility into your people and processes so you can optimize activities to meet business objectives.

Process Intelligence leverages iHub so users can easily access the insight they need when they need it. It provides pre-built reports that can even be customized to include dimensions specific to the application built in AppWorks. This eliminates the complex task of building business intelligence reports, which is typically performed by a data scientist.

## Magellan

Through OpenText™ Magellan™, AppWorks provides applications with a pathway to AI, machine learning, task automation and predictive analytics.

Massive amounts of structured and unstructured data from any source can be leveraged and combined with AppWorks application data to enable machine-assisted decision-making, automation and business optimization.

As Magellan comes up with findings and insights, it will deliver them through iHub reports and dashboards to stakeholders in a visually compelling way that makes them easy to understand and act on. With AppWorks' iHub panel, these visualizations can be brought back into the AppWorks application to have the insights where they need to be, with the decision makers and knowledge workers.

Magellan can solve requirements specific to any solution built in AppWorks and digest case management archives, employee records and similar unstructured documents to help find more efficient processes, maintain SLAs, accurately rate employee effectiveness, manage risk prudently or detect fraud, just to name a few examples.

## Content storage considerations

A developer does not have to worry about whether their application will run in the cloud or on-premises. They only need to build it once and can deploy it where they want—with no extra steps. They only need to focus on how the application will work and what document store (Content Suite, Core, xECM) to use.

## Application deployment

When a developer completes an application, it will be initially deployed on the non-production instance of the infrastructure for testing. Once approved, it will be moved to the production instance. At that point, they can install, set up and commission the application in their cloud infrastructure.

## Conclusion

As organizations continue their digital transformation, developers face a lot of pressure to do more, faster. The low-code application development platform will ensure developers thrive under pressure. Now they can use a consistent look and feel and infrastructure and consistent communication methods to build applications. With these capabilities, they can more quickly create applications and more easily deliver features that help business users succeed.

## About OpenText

OpenText, The Information Company™, enables organizations to gain insight through market leading information management solutions, on-premises or in the cloud. For more information about OpenText (NASDAQ: OTEX, TSX: OTEX) visit: [opentext.com](https://opentext.com).

## Connect with us:

- [OpenText CEO Mark Barrenechea's blog](#)
- [Twitter](#) | [LinkedIn](#) | [Facebook](#)

## Appendix A - Data domain models and properties

The sections in this appendix contain information on the following domain models:

- **Identity component** – Provided out of the box with AppWorks, it is automatically installed and deployed and includes artifacts for the developer to leverage in their applications
- **Case Management Application Accelerator** – Available to developers to manually make available and extend in their applications
- This appendix provides an overview of each domain model with a brief description of their use. Detailed information on all properties of each entity shown in the following domain.

### Identity component

#### Domain model overview

Most of the enterprise applications require common entities such as Person, User and Enterprise. The purpose of this component is to provide such entities as part of AppWorks itself so that multiple applications can use the same data set, avoiding the data duplication for such entities in each application.

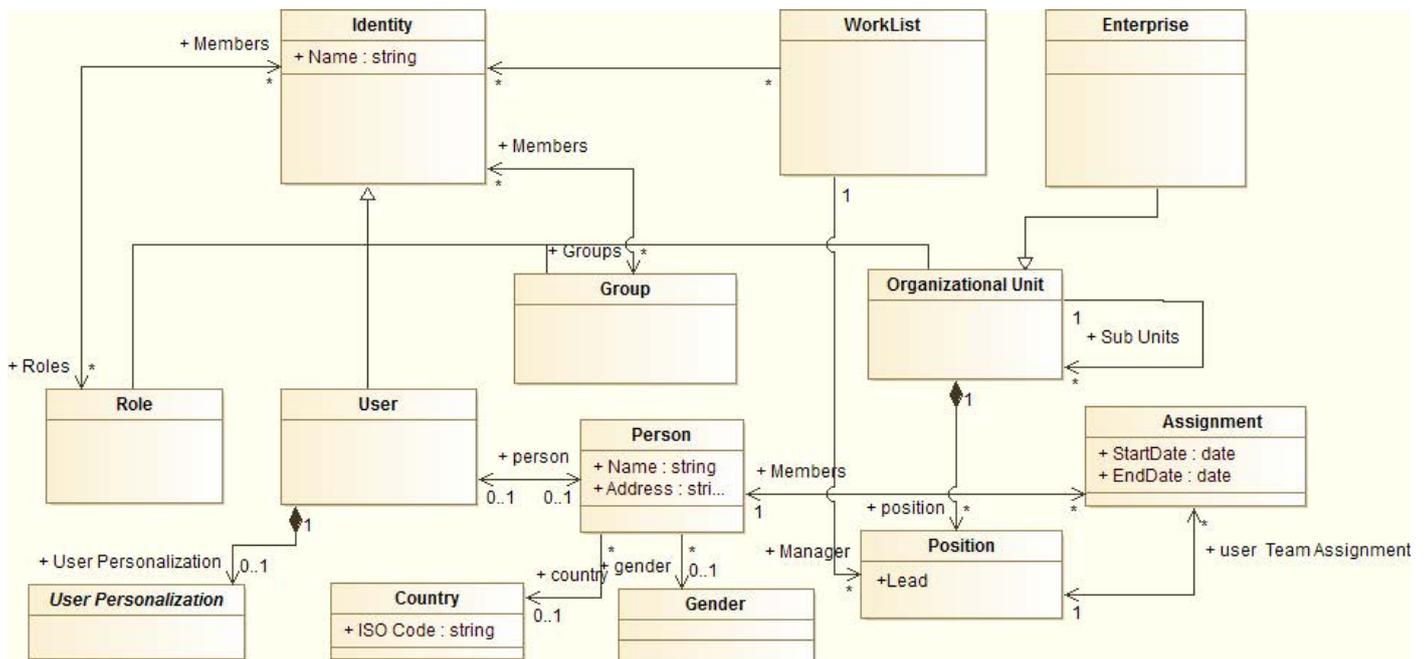
Apart from basic Entities like Person, User, Enterprise, Address, State and Country, the Identity component provides entities to store organization structure-related information, such as Organizational Units, Groups and Roles.

The Identity component also provides integration with OpenText Directory Services for synchronizing user information and User/Group assignments in real time into Identity component entities.

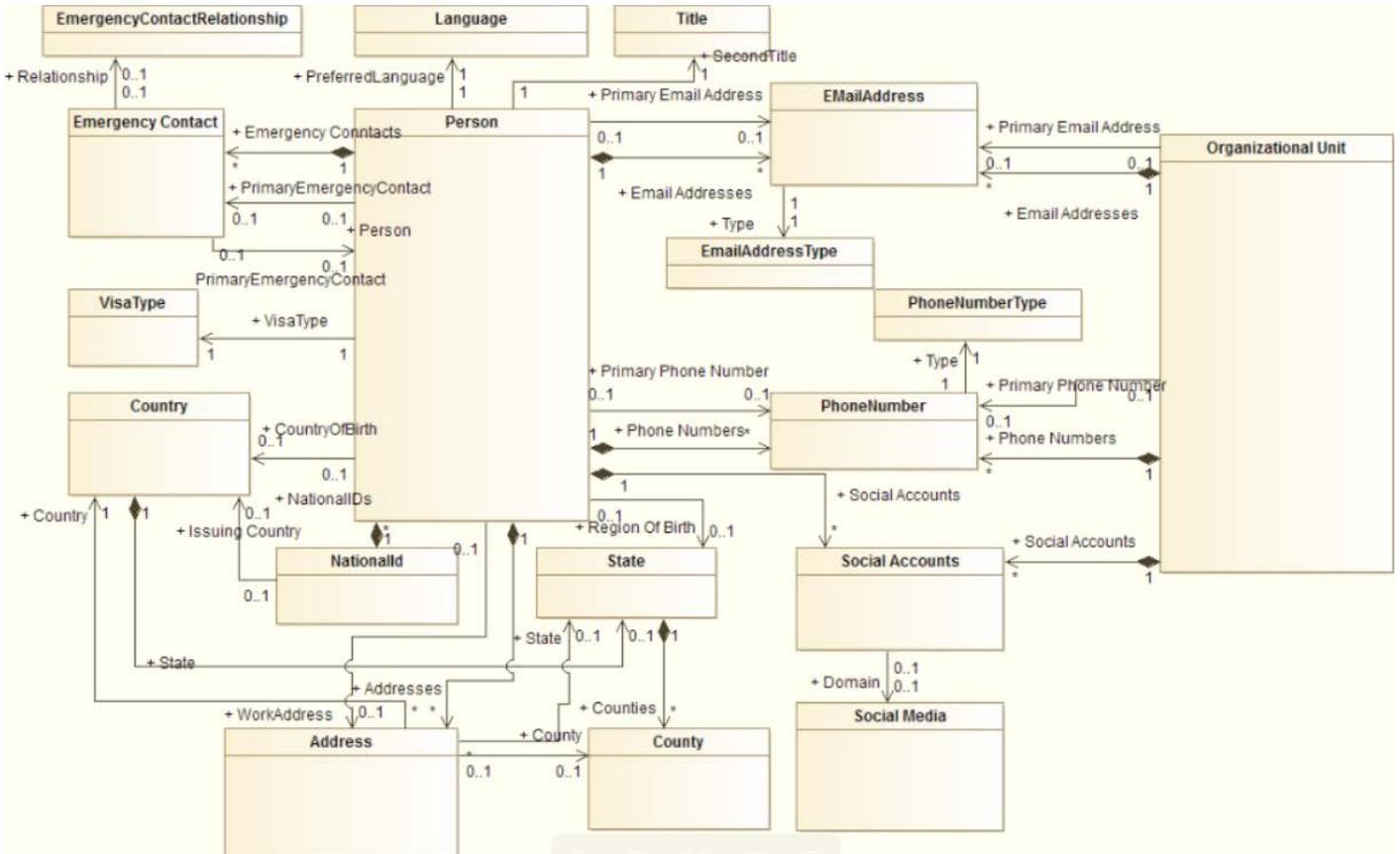
This package also provides APIs to access/update Personal and Organization structure related information.

#### Domain model diagram

The Identity component domain model is split into two diagrams, one showing overall domain model (without details on Person entity) and another one showing more detailed diagram around Person entity.



Detailed domain model of Person entity

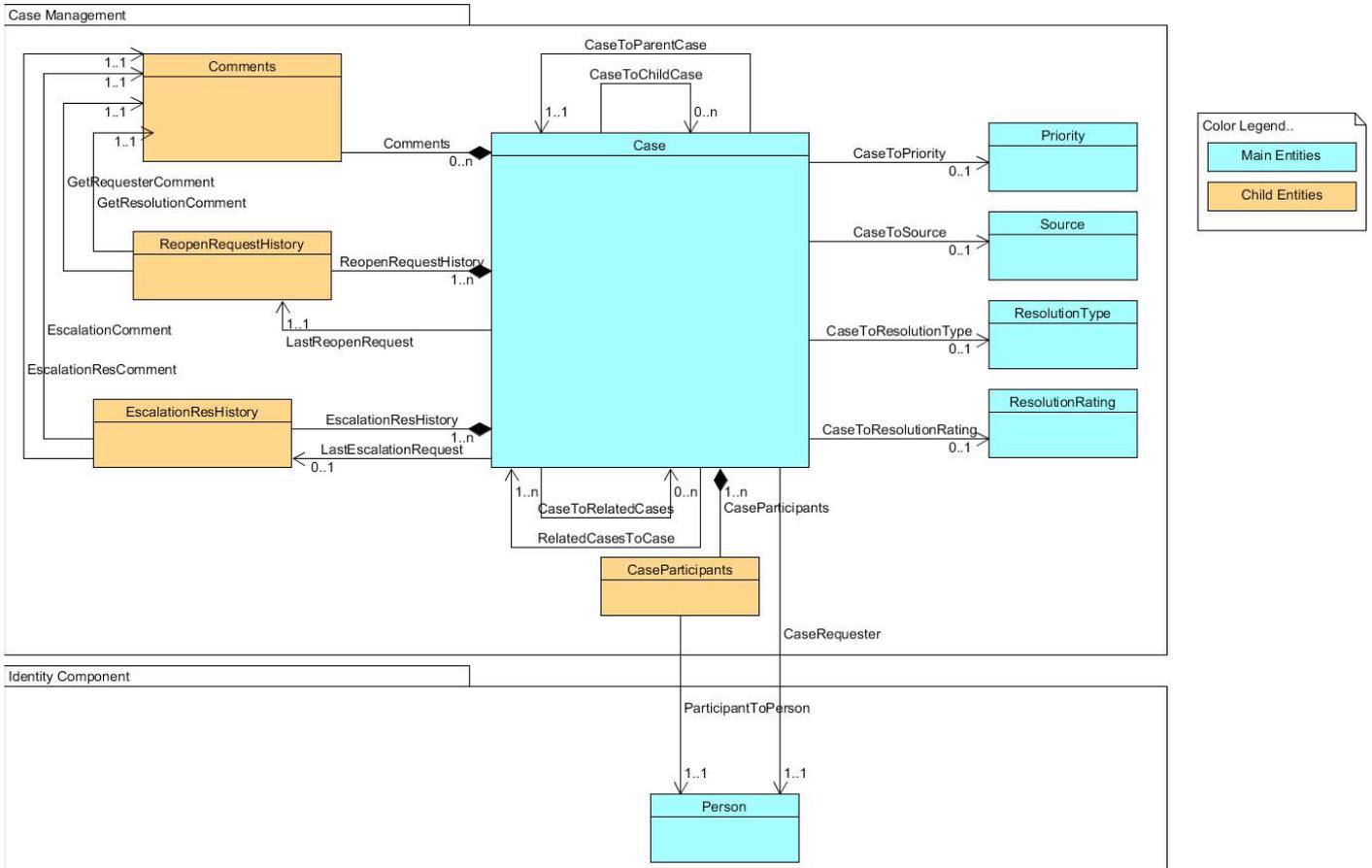


Case Management Application Accelerator

The Case Management Application Accelerator provides case management capabilities that developers can extend to create a robust case management application. It also includes knowledge management capabilities with the Help Topic data model. This allows application developers to create their own application help to assist users.

Domain model overview

The Case Management application can automate a complex, unpredictable and unstructured business process. It empowers knowledge workers to handle scenarios with dynamic and unpredictable interactions between people, processes and content. Case Management offers knowledge workers complete flexibility to intervene and decide the next course of action in the process, to modify case progression and to initiate ad-hoc jobs based on the tasks at hand. Case Management offers organizations flexibility to adapt processes according to their business requirements. After a few iterations, organizations can further customize and configure this generic system by means of enhancing or subtyping.



**Help Topic domain model overview**

The Help Topic Management component provides capabilities to manage Help articles in various formats, for example, File, Text and Hyperlink to an existing webpage. This article can be categorized and searched based on the keywords associated with the articles. These modules also provide search and view capabilities, which other applications can use to find content more quickly and easily and to help users find enough information for a specific category or keyword.

