Creatio Academy

App development in Creatio

Version 8.0

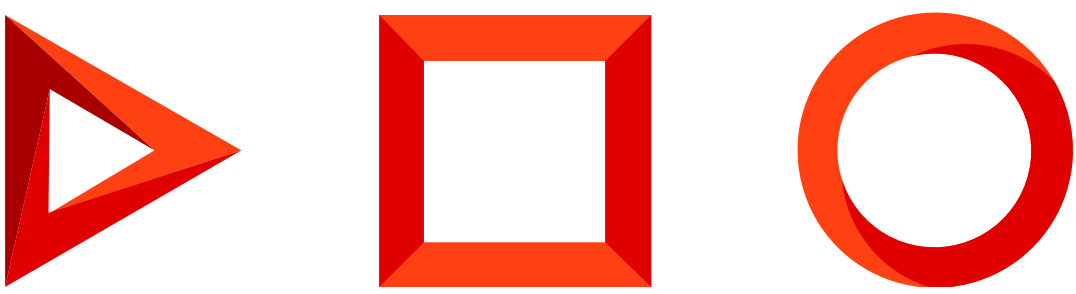


Table of Contents

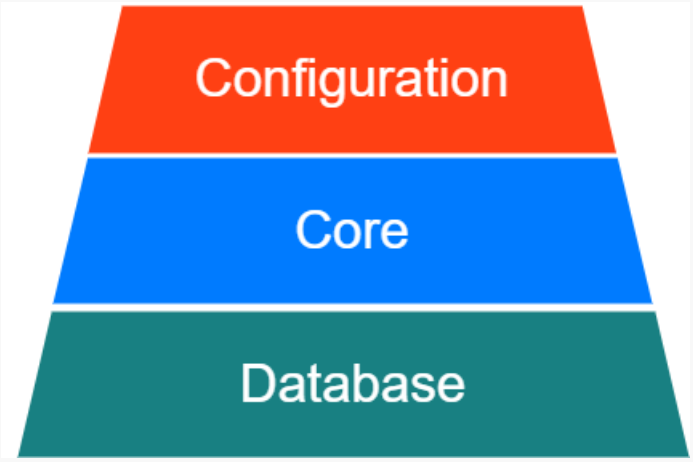
Creating applications on Creatio platform

Beginner

Creatio is a low-code platform designed to accelerate application development, implementation, and scaling. The platform is built with customization, flexibility, and scalability in mind. This makes application development possible for developers of varying skill levels – from a business analyst to a full-stack developer. Application development in Creatio allows for different levels of customization depending on the complexity and type of business goals.

Creatio customization levels

Creatio architecture comprises several logical levels of interaction:



Note that the Creatio core level is an unmodifiable component. Development in Creatio is done on the configuration and database levels.

Database

Database is the physical data storage level. The database stores client data, application settings, and access permissions.

Creatio tools let users work with data in the UI. As such, you do not need to work with database objects directly.

For certain tasks, the database level development is the most logical solution and the fastest method. You can implement custom business logic on the database level using views and stored procedures. Afterwards, you can call the business logic from custom configuration elements.

Core

The **core** is an unmodifiable part of Creatio. It is a set of libraries implementing the base Creatio functionality.

The **back-end libraries** are written in C# with the .NET Framework platform classes. Developers can create back-end class instances and use the the back-end libraries, but they cannot make any changes to these classes

and libraries.

The main **back-end core components**:

- ORM data model and its operation methods. In most cases, we recommend using the object model, though the back-end core components also allow for direct access to the database.
- Packages and replacement mechanism.
- Creatio web services.
- The main designer and Creatio section functionality.
- Third-party service integration libraries.
- Business process service (`ProcessEngineService.svc`). This Creatio element executes algorithms depicted as flowcharts.

The **front-end core classes** are written in JavaScript on top of different frameworks. Developers can use these classes to create UI and implement other browser-side business goals. The main **purpose** of the front-end core components is to manage the operation of client modules.

The main **front-end core components**:

- External libraries of client frameworks.
- Sandbox – a special client core component that manages the message exchange between client modules.
- Base modules – JavaScript files that implement the functionality of the primary Creatio objects.

Configuration

The **configuration** is functionality available for Creatio users of a specific workspace, namely:

- Server logic.
- Classes generated by Creatio settings automatically.
- Client logic (pages, buttons, actions, reports, business processes, and other configuration elements).

The configuration is an easily modifiable part of Creatio. A specific configuration constitutes the following element types:

- **Objects** – data-storing entities that combine a table in the database and a class on the server's end.
- **Business processes** – configurable elements that depict a user action flowchart.
- **Client modules**.
- **Server modules**.

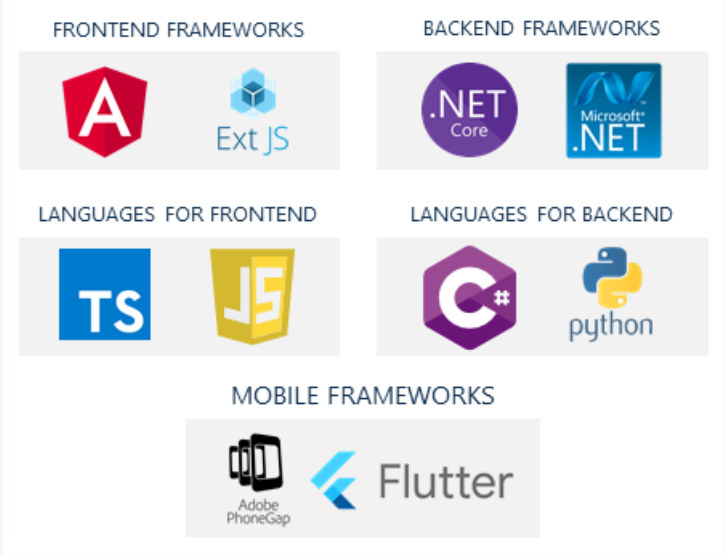
Application development tools

Creatio provides a wide variety of tools to create new and modify existing applications.

Open source technologies

Support of standard programming languages and frameworks streamlines the development and troubleshooting, as well as makes it easier to find qualified specialists with the necessary technology stack.

Creatio supports the following **technologies**:



The **built-in IDE** lets you implement complex business logic, integrations, and customizations, accelerating many common configuration procedures. Use C# and JavaScript to solve the following **problems**:

- Expanding and modifying Creatio functions.
- Organizing the interaction with version control systems.
- Migrating changes between the development, testing, and production environments.

Creatio supports **third-party IDEs** (e.g., Microsoft Visual Studio, MS Code, Rider) that let developers work with projects in a local file system. The familiarity of the IDE, as well as the variety of plug-ins, extensions, and version control system integrations, etc., streamline the development. Developers can save time spent on learning new tools and concentrate on coding.

Low-code/no-code development

The low-code/no-code development **tools** are a set of drag-and-drop UI editors. They let you solve the following **problems**:

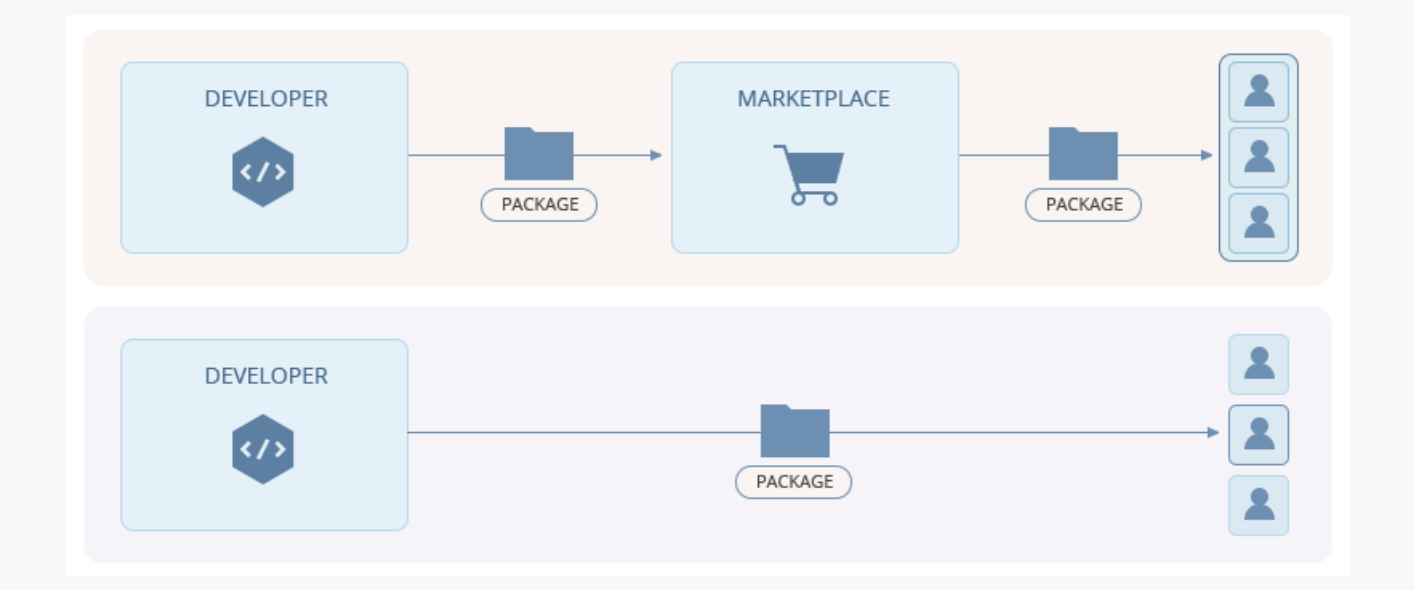
- Business process and dynamic case automation.
- Data structure modeling.
- UI modeling for web and mobile apps.
- Dashboard creation and report building.
- Integration setup.
- Predictive model building, etc.

The low-code tools cover most use cases. Minimal coding, as well as easily accessible built-in tools, let business experts, power users, analysts, and citizen developers. Learn more about no-code tools in the No-code article.

Package mechanism

Regardless of the tools used to customize the platform, Creatio bundles all changes into one or more packages – key Creatio architecture components. The package architecture lets you create separate modules from autonomous code blocks, as well as control the package hierarchy and versioning. That way you can expand the configuration, as well as migrate changes between the development, testing, and production environments, faster. The Marketplace solutions are based on the same mechanism.

Any Creatio product is a set of packages installed on top of the Creatio core. A package contains object schemas, the source code, business processes, reports, etc. It also may contain third-party assemblies, SQL scripts, system settings, and custom data.



Creatio extension model is based on the **open-closed principle**, where major application logic is closed for direct manipulations but open for extensions with custom packages. A package from another publisher (an integrator partner, a Marketplace developer, or a customer) can expand any package. This allows the platform to combine out-of-the box products, marketplace solutions, and client customizations effectively in almost any variation.

The **package architecture** is the main way to deliver and deploy custom applications, extensions, and templates, seamlessly integrated with Creatio Marketplace. Creatio Marketplace is an ecosystem for developing, distributing, and acquiring customizations – from custom apps and templates to updates and changes to industry applications.

Packages do not affect the core platform logic, allowing for smooth delivery of updates to Creatio Core, as well as parallel deployment and updates to generic functions.

DBMS–independent architecture and Creatio's ORM

Creatio is a **DBMS-independent platform** based on its own ORM. This allows the developers to build and deliver custom apps to various Oracle, PostgreSQL, or MS SQL Server configurations easily without any changes to the codebase.

Process-based business logic

This is an environment where full-stack developers and business analysts interact. Users can create their own

business logic by simply designing the needed processes in a drag-and-drop business process editor. Learn more about working with business processes in the BPM tools article block.

Integrations

Creatio provides all necessary tools for third-party system and app integration, including the support of the REST API, OData protocol, SOAP services, OAuth and LDAP authentication. You can develop integrations as parts of either Creatio or a third-party app. Complex tools ensure data security both during the integration for identification and access control, and during the user structure management.

No-code

Beginner

Low-code and no-code technologies use visual interfaces to enable users to develop their own IT solutions without in-depth knowledge of any programming language.

No-code technologies cater to citizen developers since they do not require knowledge of the programming language syntaxes. Organizations choose platforms that combine both technologies for more flexibility and control over the software development life cycle.

Advantages of low-code platforms:

- Fast application development.
- Fast deployment.
- Application execution and management.
- Declarative high-level development.
- Ability to model data, develop interfaces, and business logic.
- User (no-code) application configuration.

Now-code tools

The Creatio platform has a wide range of low-code/no-code customizations: from customizing the existing applications to creating the user's business solutions.

Drag and drop visual designers enable the users of low-code platforms to set up applications for solving a multitude of business tasks: from automating customer processes and enhancing teamwork to data management and third-party integration.

Process designer

The process designer is a visual designer for building "executable" business processes with BPMN 2.0 notation. Executable business processes enable the implementation of custom business logic, from automating routine tasks to creating complex iterations. Employ ready-to-use elements to design business processes that enable users to plan their activities, work with interface pages, process data, call web services, and more. The user-friendly setup interface and built-in validation tools will help not just to design or update a business process scheme, but also to debug the scheme accounting for all execution details and nuances. Use the process designer to automate and describe business processes of varying complexity for better performance.

The principles of working with business processes in Creatio are provided in the business process documentation.

Case designer

Case management enables users to automate unstructured processes with a dynamic flow in line with the established business logic. A business case consists of stages. Each stage may include a sequence of consecutive or simultaneous “steps,” i.e. automatic or manual actions. You can use the no-code case designer to change the sequence of steps in a process stage, move steps to other stages, or rearrange the sequence of stages using the drag&drop designer.

Learn more about working with business cases in the article.

Section wizard

Create and set up sections, pages, and mini pages to add or edit section records fast. Use the section wizard to add new sections and edit existing ones. Use the visual designer to change the positions of fields, add or hide fields, tabs, and details. Employ the user interface to change the business logic of the system. For example, set up a conditional view or change which section fields are required or available.

Learn more about the section wizard in the article.

Machine Learning technology

The machine learning technology enables decision-making automation by analyzing historical data and recognizing correlations between large amounts of data. For example, you can use Creatio smart technologies to set up customer profile categorization, route help desk calls, predict the probability of closing a sale, or recommend products to your customers. Use the machine learning algorithms and other AI technologies to accelerate data processing, reduce the number of manual operations, and improve the quality of decision-making.

Learn more about working with the machine learning service in the article.

Built-in integration tools

The advanced integration potential (based on .Net, REST, SOAP, OData, open API, and other tools), as well as a powerful administration and access control system, accelerate the safe integration of Creatio into the digital ecosystem of any enterprise. The Creatio platform also enables unlimited third-party integration flexibility.

Integration options using low-code technologies are covered in the article.

Back-end (C#)

 Beginner

Creatio solution development involves several customization levels depending on your business needs. That said, the core level is an unmodifiable system component, and Creatio development is done on the configuration level.

Back-end level Creatio customization workloads involve working with data, web services, setting up the objects' business logic, etc.

Creatio offers a set of various tools that fulfill these business needs — from developing the [*Source Code*] or

[*User Task*] schemas in built-in IDE to creating completely custom projects connected to Creatio as external libraries.

Back-end development workloads

The back-end level involves the following workloads:

- The ORM data model and the methods of working with it.
- Implementing direct access to the database.
- Creating and using Creatio web services.
- Setting up integrations with external services.
- Working with Creatio components and microservices
- Extended setup of business processes and built-in Creatio objects' processes.
- Developing the objects' business logic.

The ORM data model and the direct access to the database

The **ORM data model** in Creatio involves `Terrasoft.Core.Entities` namespace classes:

- `Terrasoft.Core.Entities.EntitySchemaQuery` — builds queries that select database table records, affected by the current user's permissions.
- `Terrasoft.Core.Entities.Entity` — accesses an object that represents a record in the database table.

The back-end core components allow for direct access to the database. However, in most cases we recommend using the object model to access data.

Creatio back-end core's class group from the `Terrasoft.Core.DB` namespace grants **direct access to the database**:

- `Terrasoft.Core.DB.Select` — builds queries that select database table records.
- `Terrasoft.Core.DB.Insert` — builds queries that insert database table records.
- `Terrasoft.Core.DB.InsertSelect` — builds queries that insert database table records.
- `Terrasoft.Core.DB.Update` — builds queries that update database table records.
- `Terrasoft.Core.DB.Delete` — builds queries that delete database table records.
- `Terrasoft.Core.DB.DBExecutor` — allows to build and execute complex database queries, for example, queries with several nested filters, various combinations of JOIN commands, etc.

Web services

Creatio service model includes a base set of web services that allow you to integrate Creatio with third-party applications and systems.

Creatio **system service** examples:

- odata — exchanges data with Creatio via the OData 4 protocol.
- ProcessEngineService.svc — runs Creatio business processes from third-party applications.

Besides the system services, there are **configuration web services** designed to be called from the client part of Creatio.

In addition to the base services, developers can create a **custom web service** to fulfill a particular project's specific business needs.

Third-party services

Creatio core has classes that let you integrate third-party services. For instance, `Terrasoft.Social` namespace lets you integrate various social networks.

Creatio components and microservices

Creatio core has classes that let you work with Creatio components and microservices. For instance, Creatio core's `Terrasoft.Sync` namespace provides classes for the built-in remote repository synchronization mechanism (Sync Engine). Sync Engine lets you create, update and delete `Entity` objects in Creatio based on the data from remote repositories as well as export data to remote repositories. The synchronization process relies on the `SyncAgent` class.

Business processes and built-in objects' processes

You may require back-end development to **set up complex business processes** ([*Script task*] process element) or to create custom **recurrent business process operations** ([*User Task*] configuration schema).

You can set up **built-in objects' processes** not only with no-code tools but also with back-end development. Custom coding allows for a more flexible object behavior configuration that accounts for specific events.

The objects' business logic

Creatio lets you develop the objects' business logic without using event sub-processes. To do this, simply create a class that inherits from the base `Terrasoft.Core.Entities.Events.BaseEntityEventListener` core class and decorate it with the `EntityEventListener` attribute. The attribute has to specify the name of the entity for which to execute the event subscription. Then, you can redefine the relevant events' handler methods.

Back-end development tools and utility capabilities

Source code schema

The back-end development's main capability is creating a [*Source code*] schema in the custom package.

All the schema changes made in the object designer are kept in RAM. To save the changes on the schema metadata level, you need to save the schema. To apply the changes to the database level, you need to publish the schema.

You can develop the [*Source code*] schema in the file system using an IDE of your choice.

Business process configuration

You can add specific back-end logic to a business process using custom code the [*Script task*] element

executes.

Similar operations are a common task when working with business processes in Creatio. The [*User task*] element that lets you specify the most fitting action for a particular situation serves this purpose the best.

By default, Creatio includes a wide variety of user tasks. In some situations, a new user task to execute a particular business process is required.

You can create a new user task with the [*User task*] configuration schema type. In its simple implementation the process task partially follows the [*Script task*] process element's logic. However, you can use the existing task multiple times in separate processes. If you make changes to the process task, they will be immediately applied to all processes that use it.

External libraries

You can add custom-made external libraries to the custom package's structure. This allows you to implement complex logic, inheritance mechanisms, and encapsulations when developing a specific project solution.

Project package

The project package is one of the Creatio tools that accelerate Creatio back-end development. This is a package that lets you develop functionality similar to a regular C# project. The new functionality is added to the file content package as a compiled library and a collection of *.cs files. When starting or restarting Creatio, Creatio will collect information on pre-set libraries in the packages and immediately load them.

Front-end (JS)

 Beginner

The front-end platform is a set of modules that are defined and loaded asynchronously on demand.

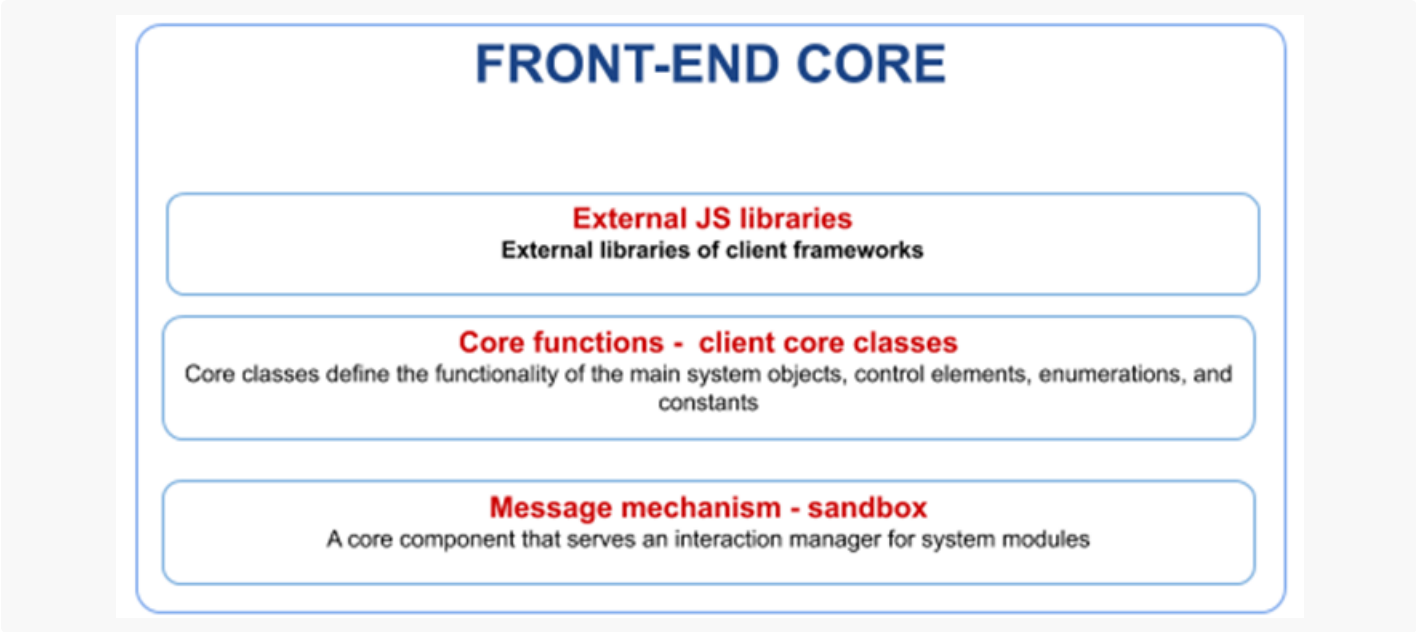
The core is the basis of Creatio's front-end.

The core level provides:

- the ability to use the object-oriented approach and inheritance technique
- the tools to define and asynchronously load modules and their dependencies
- the functionality of base Creatio elements
- the module communication mechanism

Front-end development in Creatio is done on the configuration level and comes down to creating new and expanding base visual modules, as well as non-visual modules, view models.

Creatio's front-end core components



External JS libraries

ExtJS — a JavaScript framework used for web application and UI development. Creatio uses ExtJS as a mechanism to establish the class structure of the core's client segment. ExtJS enables an object-oriented approach, which is not implemented in pure JavaScript. Use it to establish classes, implement an inheritance hierarchy, and group classes into namespaces.

RequireJS — a library that implements the Asynchronous Module Definition (AMD) approach. The AMD approach declares the mechanism for the definition and asynchronous loading of modules and their dependencies.

Angular — a JavaScript framework for single-page application development. Use it to expand browser applications based on the MVC template as well as to streamline testing and development. Creatio supports embedding custom Angular components using a common Angular core.

Base JS classes

An unmodifiable part of Creatio. These classes:

- Ensure the operation of the client configuration modules.
- Define the functionality of base Creatio objects, controls, lists, and constants.
- Are stored as executable files in Creatio folders on the drive.

Messaging

Sandbox — the core component that manages the interaction of Creatio modules. The sandbox lets Creatio exchange data between the modules (`sandbox.publish()` and `sandbox.subscribe()` methods) and load the modules into the Creatio UI on demand (`sandbox.load()` module).

Asynchronous module definition

The front-end platform has a **modular structure**.

Module — a functionality set encapsulated in a separate block that can also use other modules as dependencies.

Module creation in JavaScript is determined by the “Module” design pattern. A classic example of this pattern's implementation is using anonymous functions that return a specific value (an object, a function, etc.) associated with the module. The module's value is then exported to the global object.

To manage a large number of modules, Creatio loads modules and their dependencies according to the **Asynchronous Module Definition** (AMD) approach.

The AMD approach declares the mechanism that defines modules and loads them alongside their dependencies asynchronously. This mechanism allows Creatio to load only the data that is necessary for a particular task. Creatio uses **RequireJS** loader for working with modules.

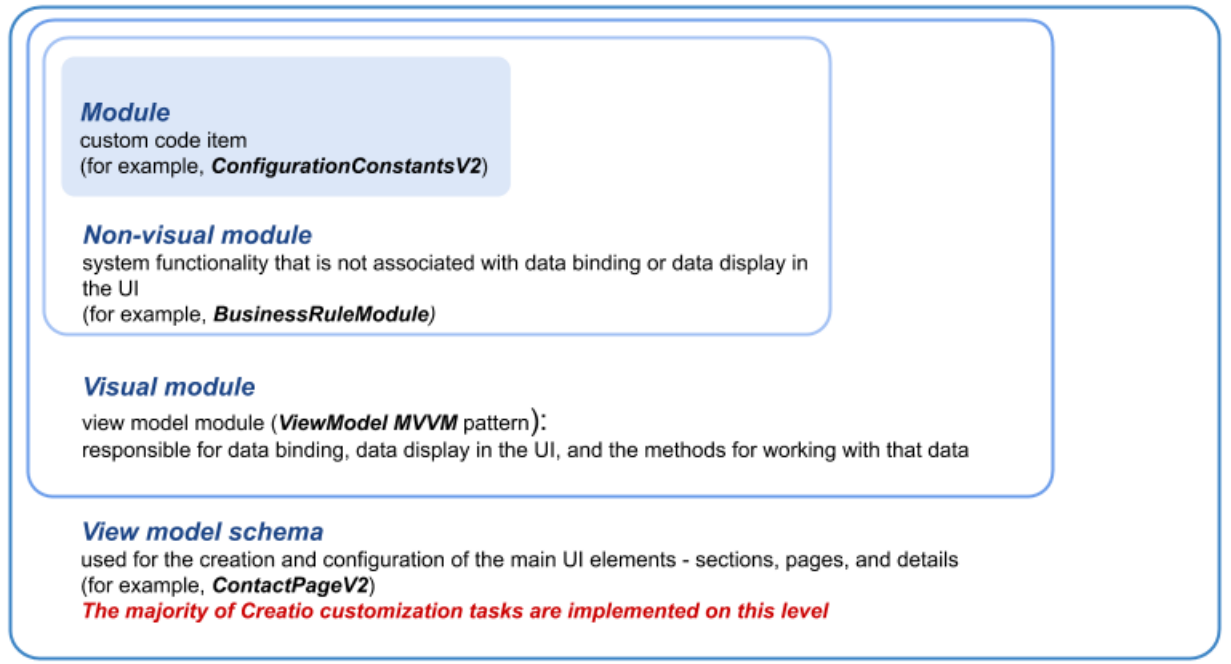
Module definition **principles** in Creatio:

- The module is defined by a special `define()` function that registers a function factory for the module's instantiation. However, this function does not immediately load the module when called.
- The module's dependencies are passed as a string value array, not via the properties of the global object.
- The loader loads all dependency modules passed as the arguments to `define()`. The modules load asynchronously. The loader determines their actual load order arbitrarily.
- Once the loader loads all specified dependencies of the module, the function factory that returns the module's value is called. Loaded dependency modules are passed as arguments to the function factory.

Modular development in Creatio

Creatio uses **client modules** to implement the entirety of the custom functionality.

The module hierarchy in Creatio



Despite some functional differences, all client modules in Creatio have the same description structure that follows the AMD module description format.

The general client module description structure

```
define(
    "ModuleName",
    "dependencies",
    function(dependencies) {
        // someMethods…
        return { moduleObject };
})
;
```

- `ModuleName` — the module's name
- `dependencies` — dependency modules whose functionality the current module can use
- `moduleObject` — the module's configuration object

Creatio has several **types** of client modules.

- The non-visual module
- The visual module
- The view model scheme

The non-visual module

This module implements Creatio functionality that is not associated with data binding and displaying data in the UI.

The non-visual module's description structure

```
define(
    "ModuleName",
    "dependencies",
    function(dependencies) {
        // Methods that implement the necessary business logic.
        return;
});
```

Non-visual module examples: utility modules that fulfill service functions.

The visual module

Use this module to create custom visual elements in Creatio.

The view model follows the MVVM template. The visual module encapsulates data displayed in the GUI's controls as well as the methods for working with such data.

It has the following **methods**:

- `init()` — the module initialization method.
  Manages the initialization of the class object's properties as well as the message subscription.
- `render(renderTo)` — the module view's DOM rendering method.
  Returns the view.

  This method only accepts the `renderTo` argument that specifies the element where the module object's view will be embedded.
- `destroy()` — the method that manages the deletion of the module's model or the view model, the unsubscription from the previously subscribed messages, and the destruction of the module class's object.

You can use the base core classes to create a visual module. For instance, you can create a class that inherits from `Terrasoft.configuration.BaseModule` or `Terrasoft.configuration.BaseSchemaModule` in the module. These base classes already have a minimal implementation of the essential visual module methods — `init()`, `render(renderTo)` and `destroy()`.

The structure description of a visual module that inherits from Terrasoft.BaseModule base c...

```
define("ModuleName", "dependencies", function(dependencies) {
    // The definition of the module's class.
    Ext.define("Terrasoft..configuration.className") {
        alternateClassName: "Terrasoft.className"
        extend: "Terrasoft.BaseModule",
        …
        // Properties and methods.
        …
    };
    // Creating the module's object.
    return Ext.create(Terrasoft.className)
});
```

Visual module examples: modules in charge of controls on Creatio pages.

The view model schema

The most common Creatio customization problems include creating and tweaking basic interface elements: sections, pages, details.

These elements' modules have a **pattern-based structure**. They are called view model schemas.

A view model schema is a configuration object Creatio generators `ViewGenerator` and `ViewModelGenerator` use to generate the view and the view model.

The base schema substitution mechanism is used for most problems related to Creatio customization. This includes all the schemas in pre-installed configuration packages.

The most common base view model schemas are as follows:

- `BasePageV2`
- `BaseSectionV2`

- `BaseDetailV2`

The view model schema's structure

```
define("SchemaName", "dependencies",
    function(dependencies) {
        return {
            entitySchemaName: "ExampleEntity",
            mixins: {},
            attributes: {},
            messages: {},
            methods: {},
            rules: {},
            businessRules: {},
            modules: {},
            diff:
        };
});
```

View model schema examples: page, section and detail schemes.

Integrations

Beginner

Integration is data exchange between systems with or without subsequent data processing. The **purpose** of integration is to transfer custom data automatically between apps.

Creatio has a wide range of integrations with external apps. Creatio's open API lets you implement integration solutions of any complexity.

Creatio has the following **ways to integrate**:

- Integration of external apps with Creatio.
- Creatio integration with external apps.

The choice of the integration method depends on the following **features**:

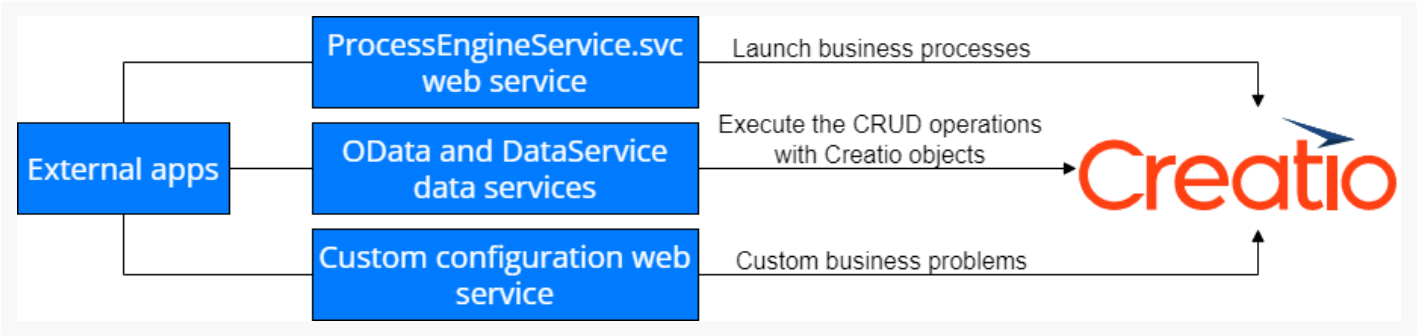
- client needs
- external app type and architecture
- developer expertise level

Integration of external apps with Creatio

The integration of external apps with Creatio solves the following **problems**:

- execute CRUD operations with Creatio objects

- launch business processes
- implement custom tasks that you can solve using Creatio's open API



We recommend using multithreaded integration to maximize Creatio throughput. Learn more in a separate article: Integration options.

Data services

The **purpose** of data services is to execute CRUD operations with Creatio objects. Learn more in separate articles: Access data directly, Data access through ORM.

Creatio lets you use the following **data services**:

- OData protocol
- DataService service

OData protocol

OData (Open Data Protocol) is an ISO/IEC-approved OASIS standard. It defines a set of best practices for building and using REST API. Use OData to create REST-based services that let you publish and edit resources using simple HTTP requests. Such resources should be identified with a URL and defined in the data model. Learn more in a separate article: OData.

Creatio supports OData 4 and OData 3 protocols. OData 4 has more features than OData 3. The main **difference** between the protocols is the data format of the server's response. Learn more about the differences between OData 3 and OData 4 protocols in the official OData documentation. Use the protocol version 4 when you integrate with Creatio via the OData protocol.

Learn more about the detailed protocol description in the official OData documentation.

DataService service

DataService (developed by Creatio) is a service that implements communication between front-end and back-end app parts. DataService lets you transfer custom data to back-end app part to be processed and saved to a database. Learn more in a separate article: DataService.

Service that runs business processes

Use the `ProcessEngineService.svc` system web service to run business processes from an external app. Learn more in a separate article: Service that runs business processes.

Custom web service

Users can create **custom web services** to solve integration problems. The configuration web service is RESTful service based on WCF technology. Learn more in a separate article: Custom web services.

Creatio integration with external apps

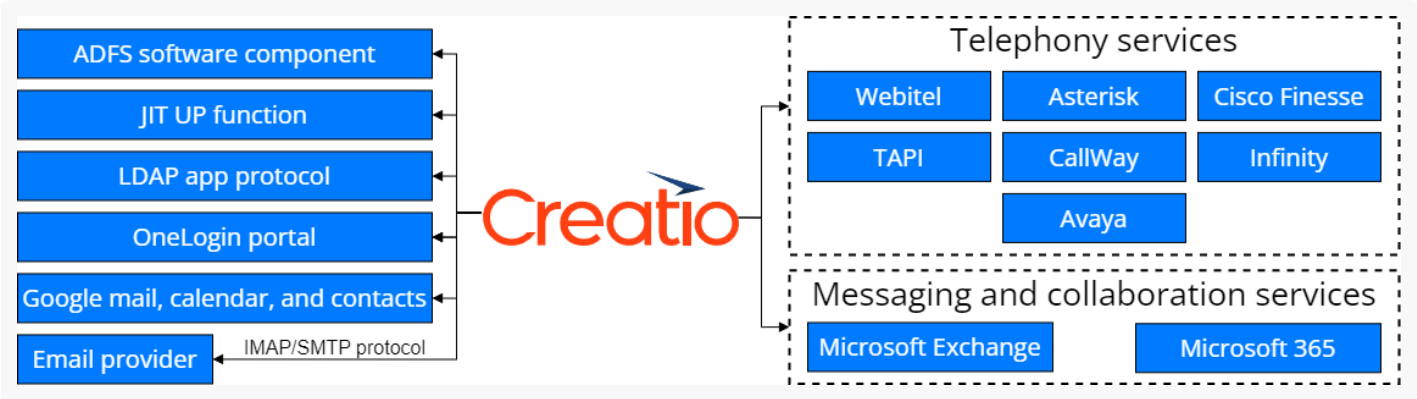
Use app tools to combine different enterprise apps into a single digital ecosystem. When you integrate Creatio with external apps, you can develop a custom solution or use an out-of-the-box integration solution.

Develop custom integration solution

Use app tools to set up integration with a custom RESTful API. You can call a web service from a business process after you set up the integration. Use REST API tools to interact with external web services with no developer involvement.

Use out-of-the-box integration solution

View the out-of-the-box integration solutions Creatio implements below.



Creatio implements out-of-the-box integration solutions with the following **apps**:

- OneLogin portal, used as a single sign-on point for all company services.
- Active Directory Federation Services (ADFS) software component to manage single sign-on for all system users.
- Just-In-Time User Provisioning (JIT UP) function, which alleviates the need to create accounts for each separate service and keep the user database up-to-date manually.
- Lightweight Directory Access Protocol (LDAP) is an app layer protocol to access a specific database that usually stores accounts of users, computers, etc.
- Mail service via IMAP/SMTP protocol.
- Google mail, calendar, and contacts.
- Webitel, Asterisk, Cisco Finesse, TAPI, CallWay, Infinity, Avaya telephony services.
- Microsoft Exchange and Microsoft 365 messaging and collaboration services.