PAYROLL ENGINE

Payroll Engine White Paper

PayrollEngine.org
Pre- Release January 28, 2026

1 Introduction

This document describes how the Payroll Engine works and the underlying concepts. Chapters 1 to 4 describe the Payroll Engine from a business perspective. Chapters 5 to 8 describe the technical aspects of the solution.

The Payroll Engine can be used to calculate company payroll data. The engine differs from conventional payroll systems in the following ways:

Case-driven	Data changes are made on a case-by-case basis and assigned to the timeline. This ensures that business data is valid at all times. Automated Payruns, including retroactive and forecast scenarios, are possible at any time.
Globalization	Business data can be settled per tenant in accordance with the respective country- and company-specific laws and local regulations.
Regulations	The payroll application is defined in regulations that build on each other in layers. There are regulations for countries, industries, and companies, such as generally binding collective bargaining agreements or regulations for pension funds.
Customer adaptations	Customizations for the customer are implemented in a separate set of regulations, enabling scalable solutions for small to large companies.
Automation	The engine has a configurable and expandable data and processing model that is defined via the REST API. The runtime behavior of the payroll application is controlled via the scripting API.

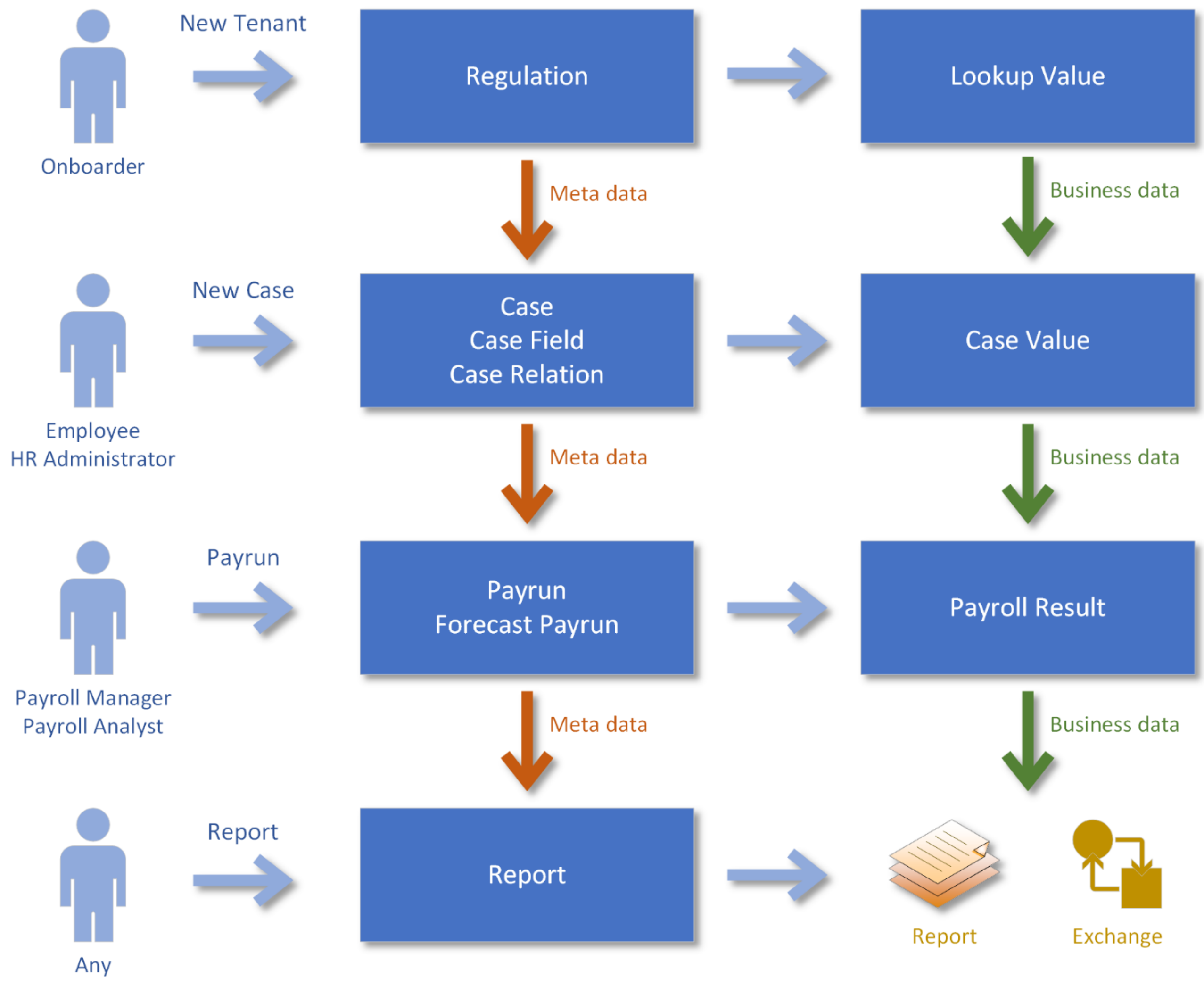
2 Payroll model

2.1 Core use cases

The central payroll use cases cover the following areas:

- Onboarding: Setting up the tenant
- Case management: Changing company and employee data
- Payrun: Cyclical and non-cyclical calculation of employee wages
- Report: Evaluation of payroll data

The following figure shows the payroll objects of the central use cases:

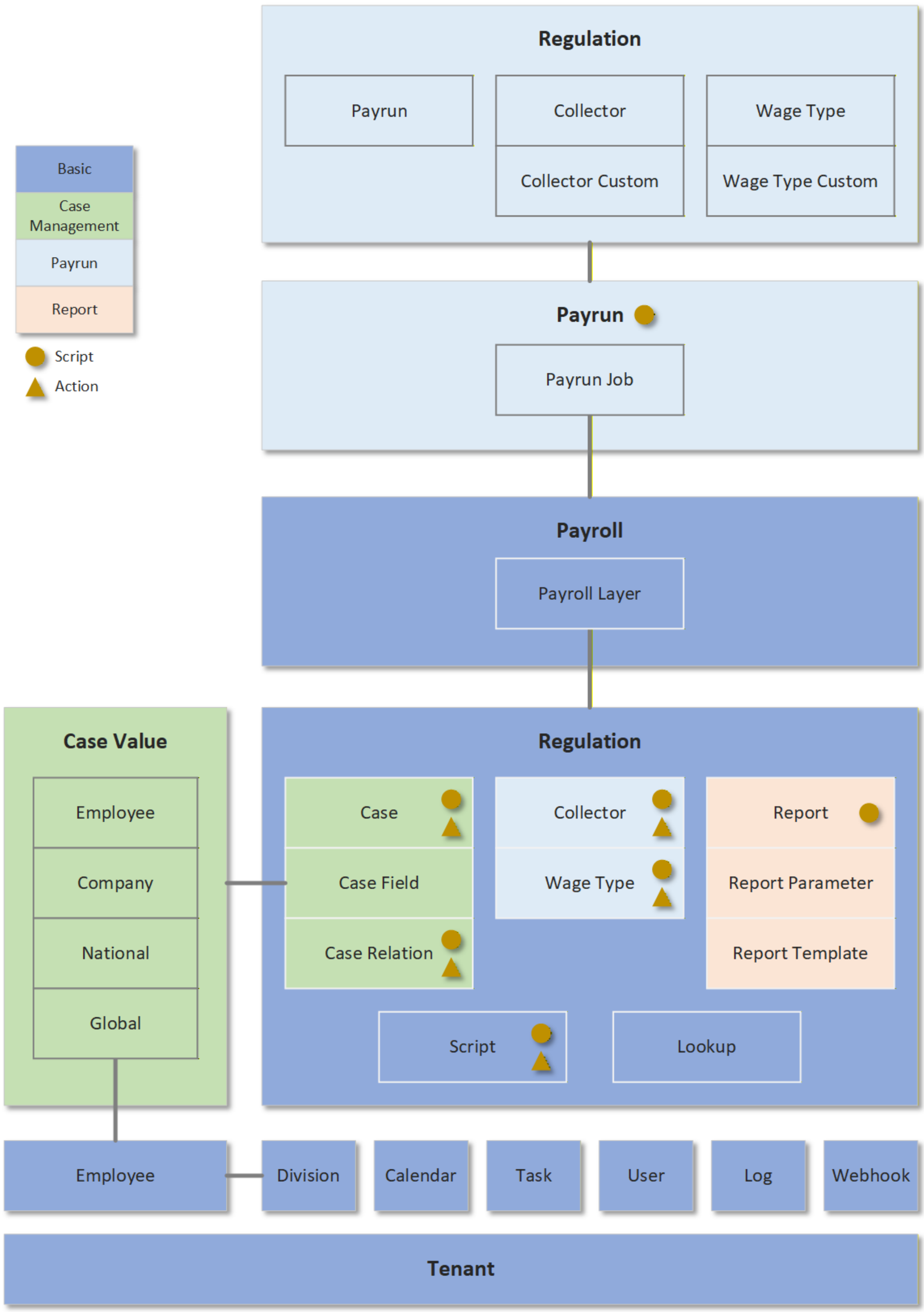


2.2 Model overview

The payroll model contains objects for the following functional areas:

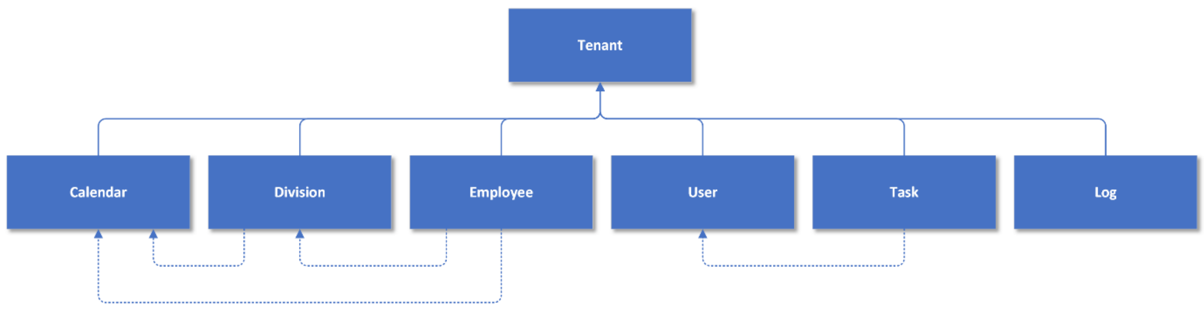
- Basic and infrastructure objects (*Basic*)
- Objects for recording business data (*Case Management)*
- Objects for processing payroll data (*Payrun*)
- Objects for transforming output data (*Report*)

The following overview shows the static topology and the control of runtime behavior with scripts and actions:



2.3 Tenant

The client-based Payroll Service offers the following objects for defining the tenant with its organizations, employees, and users:

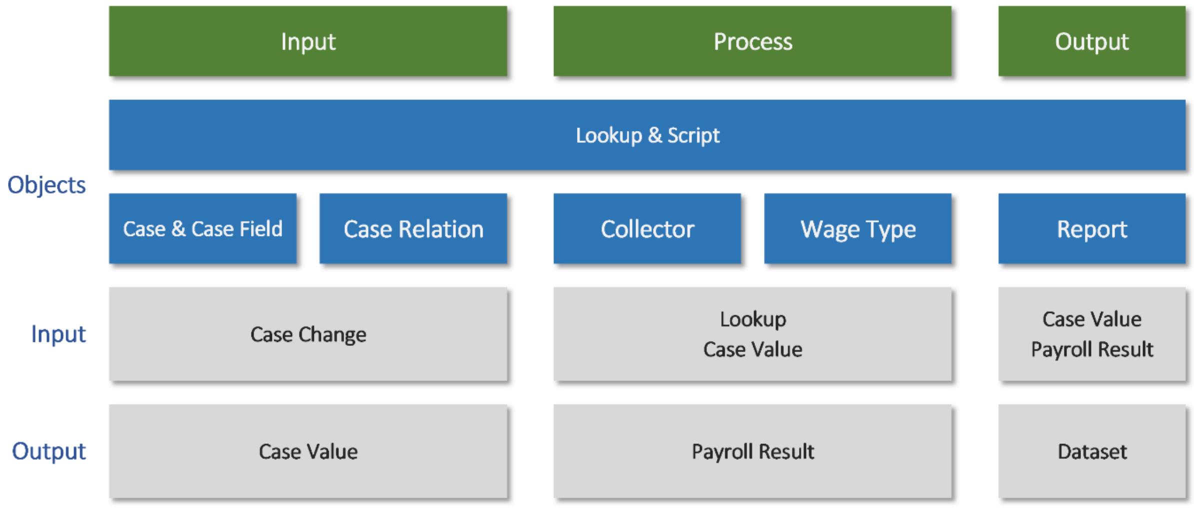


Tenant	Contains all client data. All client data refers to this object.
Calendar	The tenants' payroll calendars.
Division	Divides the company into divisions. Each tenant consists of at least one division.
Employee	Employees who are assigned to one or more divisions.
User	Users of the system.
Task	User tasks.
Log	Client-specific log entries.

With the exception of shared regulations (seeShared Regulations), all domain objects described below are assigned to the tenant.

2.4 Regulation

A regulation comprises the definition of wages with the elements of wage data entry, processing, and output:

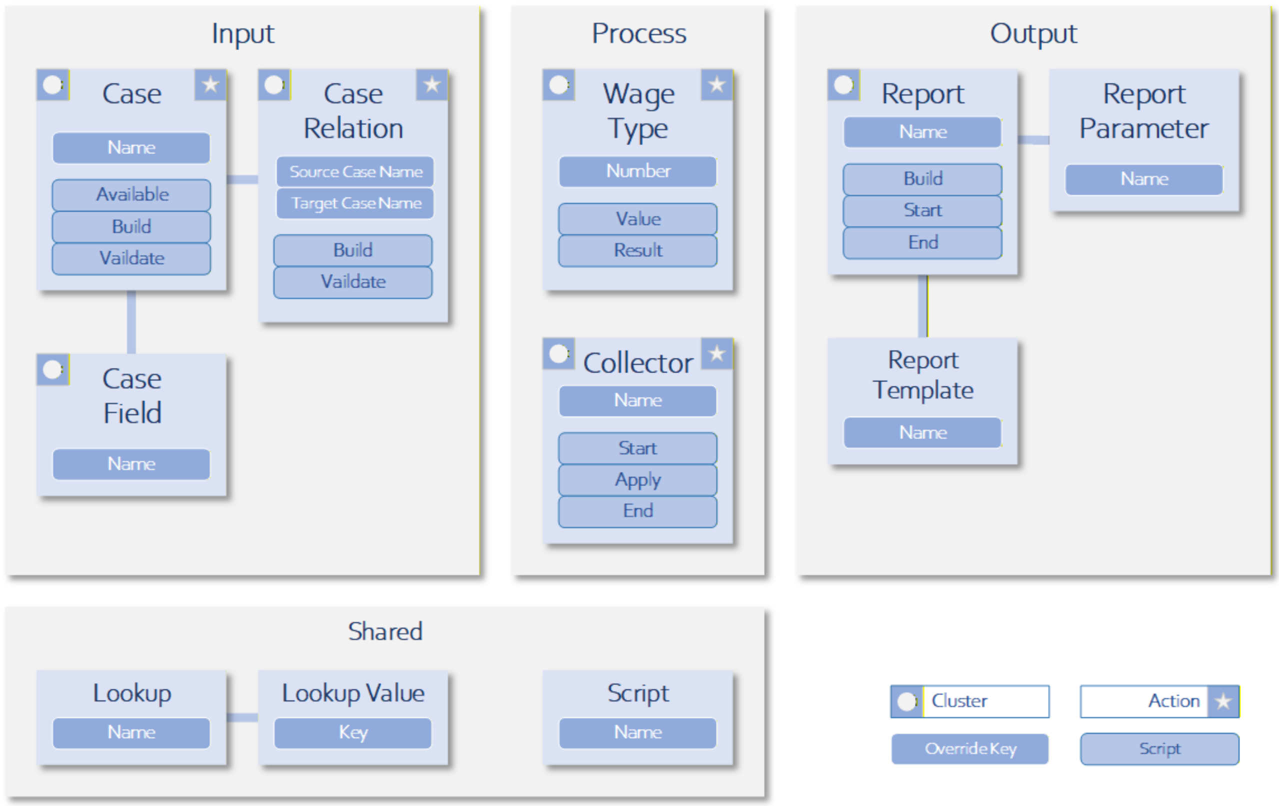


The regulations cover the following topics:

- Definition of the data model and business rules, seeCase fields
- Calculation of wage data, seePayrun
- Evaluation of company and employee data, seeReport
- Testing of company and employee data, seePayroll Tests
- Extending existing regulations, see
- Sharing software between tenants, seeShared Regulations
- Deploying software, seeDeveloping and providing regulations

2.4.1 Rule set objects

The regulation comprises the following objects:



Regulation	Contains the data of a regulation.
Script	Common code for the functions within the regulation.
Lookup **Lookup Value**	Selection data for applications (e.g., gender) and value tables for scripting functions (e.g., control tables).
Case	Wage Use case in the areas of global, national, company, and employee.
Case Field	Field that describes the value of the case. Employee data can apply to one or all Divisions.
Case relation	Regulates the dependency between two cases.

Wage Type	Determines the wage type and its calculation. Wage types are calculated in the order of the wage type number with a decimal value (e.g., 1000). A Subordinate wage types can be included in the processing order by using decimal places in the wage type number (e.g., 1000.1). The wage type determines which collectors or collector groups are relevant.
Collector	Aggregates the wage data during the Payrun, e.g., for the wage bases. The type of collector is controlled by the aggregation type, e.g., summation, minimum, maximum, counting, etc. Collectors can be executed individually or in groups.
Report	Definition of the report.
Report parameters	Parameters of the report.
Report Template	Report template in one language.

Overlays can be used to override a rule object with a higher-level industry or customer rule set (seePayroll). The object to be overridden is identified by the overlay key, for example, the case name.

Cluster objects offer an extended tagging mechanism to include or exclude similar objects according to application criteria (seePayroll Cluster s).

There is an audit trail for each regulation object (seeAudit Trail) with corresponding REST endpoints. Changes to a regulation (e.g., annual update) are managed in versions, with each version being valid from a specific accounting period. This ensures that the time data for Payruns is correct for retroactive accounting and forecasting.

2.4.2 Regulation namespace

To avoid name conflicts between regulations, each regulation is assigned a namespace. This can be a country and/or industry code. Once the namespace has been defined, it is taken into account when creating and updating regulation objects. However, this does not apply to object references in another regulation.

Namespace	US
Object name Storage	Salary US.Salary
Object name Storage	CA.Salary CA.Salary

This convention significantly improves the readability of theScripting regulation .

2.4.3 regulation inheritance

The regulations are based on the principle of inheritance, i.e., an object in a regulation can be overridden by an object in a higher-level regulation. The objects to be overwritten are identified by the override key:

• *Case*	Name (key)
• *Case Field*	Name
• *Case Relation*	Case names from source and target
• *Collector*	Name

Wage Type	Wage type number
Report	Name
Report parameters	Name
Report template	Name
Lookup	Name
Lookup Value	Key

2.4.4 Case fields

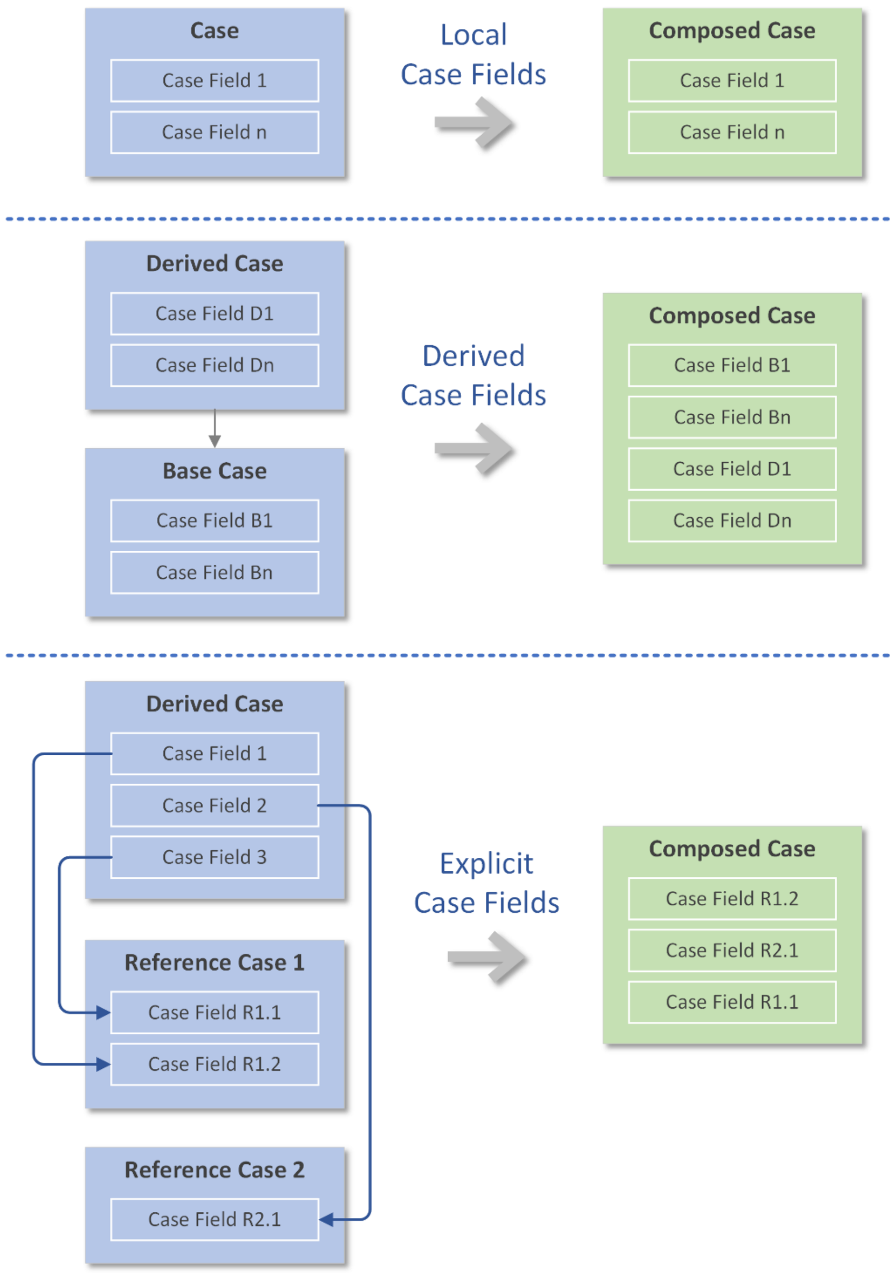
The *Case* object is used to record case data. The *Case Field* object represents an input field in the case. The *Case Relation* object defines the relationship (availability and value transfer) between two cases.

When recording the case value, its validity period is defined with the date range (from/to). This enables the mapping of complex business cases, such as the cancellation of previous changes or the scheduling of salary adjustments (seeTime types). This feature distinguishes the Payroll Engine significantly from conventional software solutions, which usually overwrite existing data.

The engine offers various approaches for compiling cases and their fields:

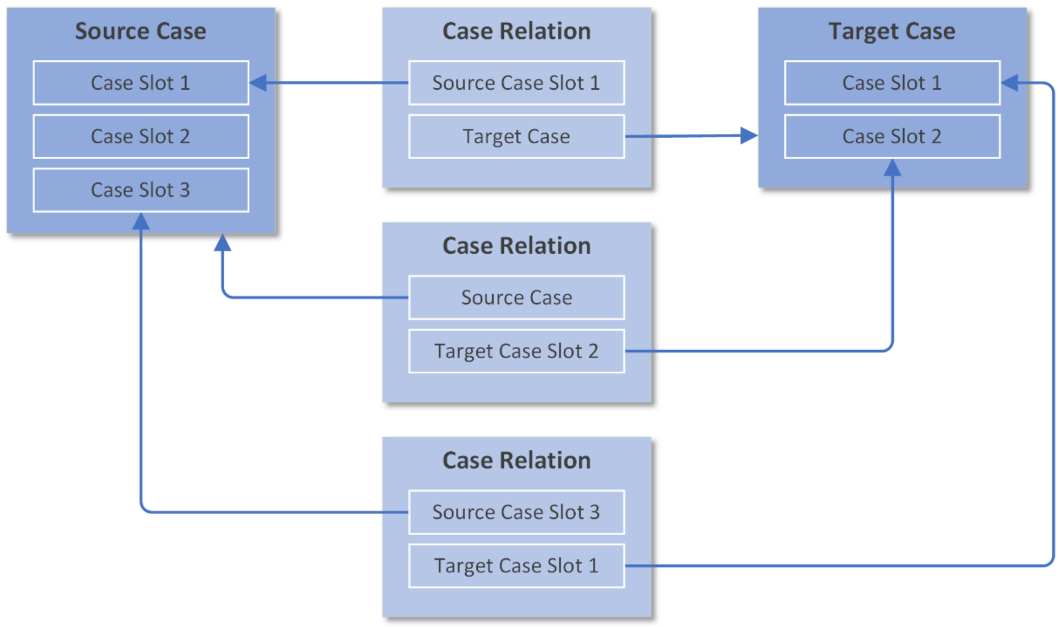
- Declaration of the case field in the case as an integral part
- Transfer of case fields from another case
- Integration of case fields from different cases

The following overview shows possible case constellations, which can also be combined:



2.4.5 Case slots

In order to be able to store multiple data records for a case (e.g., child or address), each variant is assigned to a case slot with a unique name.



2.4.6 Lookups

The lookup contains selection data from a set of Regulations for

- externally determined data from external sources, e.g., a control table
- Unchangeable selection values that do not change during a payroll period
- Values that can change cyclically or periodically

A lookup consists of a list of values (*lookup values*) that is made up of a key and a value. The key is a text that is unique within a lookup. The lookup value is described by a JSON structure. The user is responsible for ensuring that the JSON structure is structurally identical for all values in a lookup.

In addition to searching for a lookup value using the key, the lookup value can also be searched for using a value (decimal number). This can be used, for example, for income-dependent tax tables.

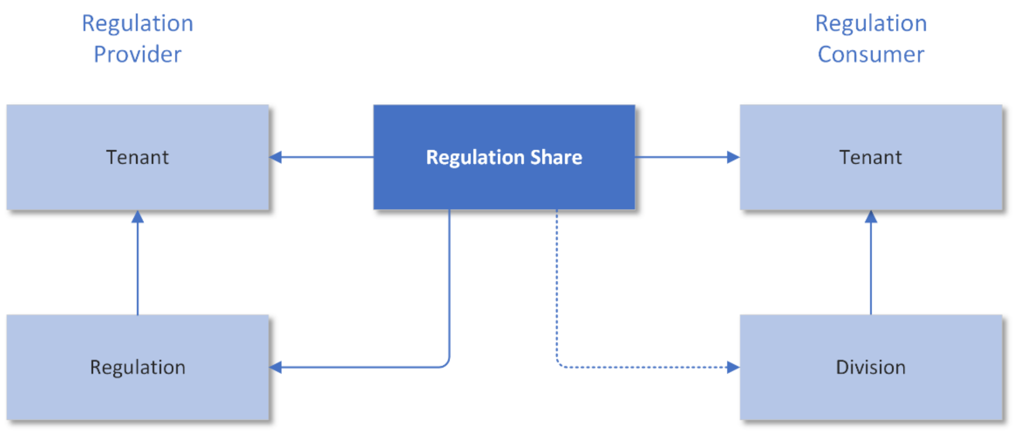
2.4.7 Application scenarios

The following areas of application are possible for payroll regulations:

- Development of customer-specific regulations in onboarding
- Service providers offer commercial Regulations
- Industry solutions, such as association solutions
- Country-specific and statutory payroll accounting
- Company regulations, local or transnational
- Regulations for testing and further development

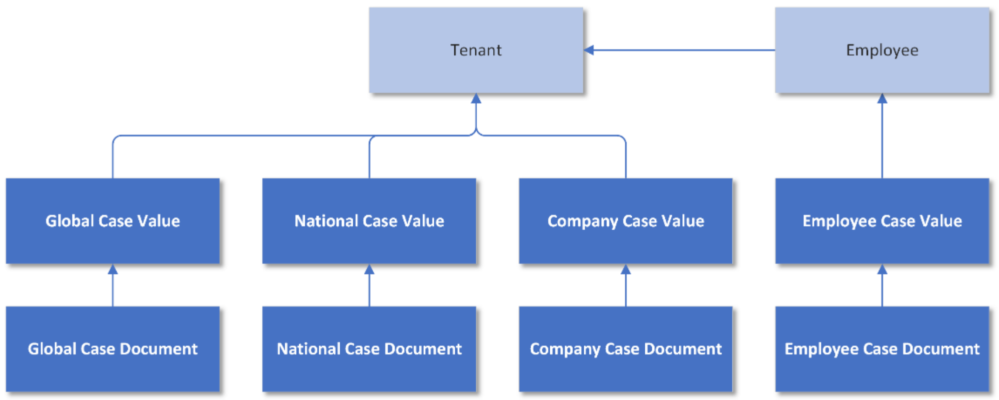
2.5 Shared Regulations

The engine offers the option of sharing a set of regulations for multiple tenants. This enables the payroll provider to centrally administer country and industry regulations. Authorizations are used to determine which set of regulations is available for which tenant (optionally, Division):



2.6 Case data

For each field of a case (*Case Field*), a value (*Case Value*) can be recorded by a case change (*Case Change*). The case data is stored physically separated by nationality, company, and employee:



Global case value	The global data.
Global Case Document	The global documents.
National Case Value	The national data.
National Case Document	The national documents.
Company Case Value	Company data.
Company Case Document	The company documents.
Employee Case Value	Employee data.
Employee Case Document	The employee documents.

A case value is available either for all or for a specific Division. It is defined in the data field (*Case Field*) of the regulation.

The case data cannot be changed and is supplemented by case changes. The data for a pay period can be determined on the basis of this change history.

The value of a case can be restricted to a validity period (start/end) (seeTime types). The payrun only takes the value into account if its period matches the payroll period.

The case types can be used in the regulations as follows:

Regulation	Global	National	Company	Employees
Tenant	✓	✓	✓	✓
Business	✓	✓	✓	✓
National	✓	✓		
Global	✓			

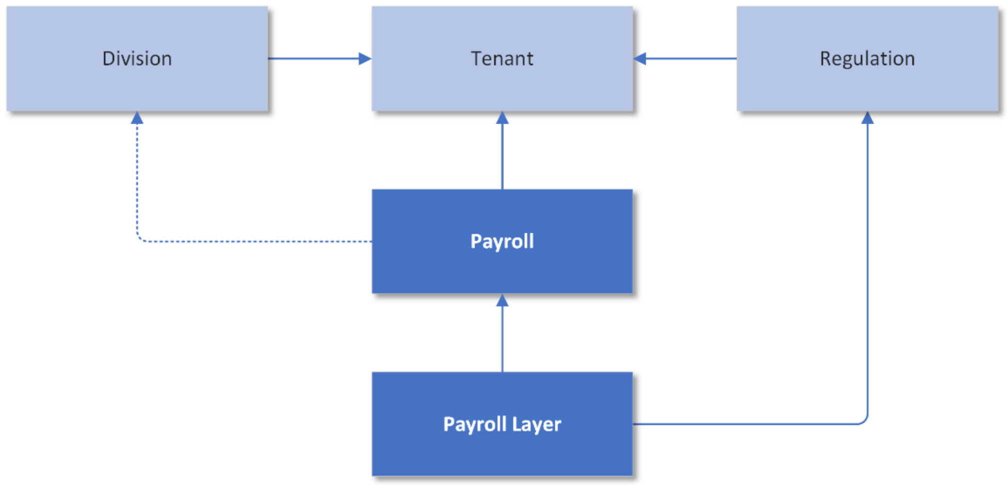
2.6.1 Cancel case data

If a case allows cancellations, the reset behavior for numeric case values is controlled by the cancellation type:

- Transfer previous value
- Initialize value to default
- Invert value
- Retain value

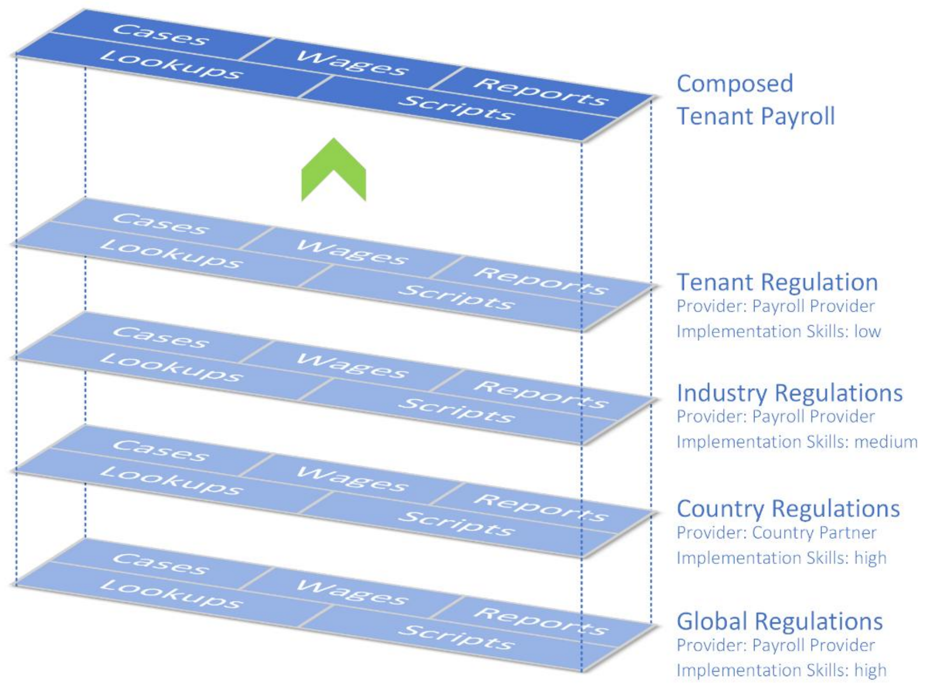
2.7 Payroll

The *Payroll* object combines several sets of regulations into a virtual payroll model using a shift model. The following objects are involved:



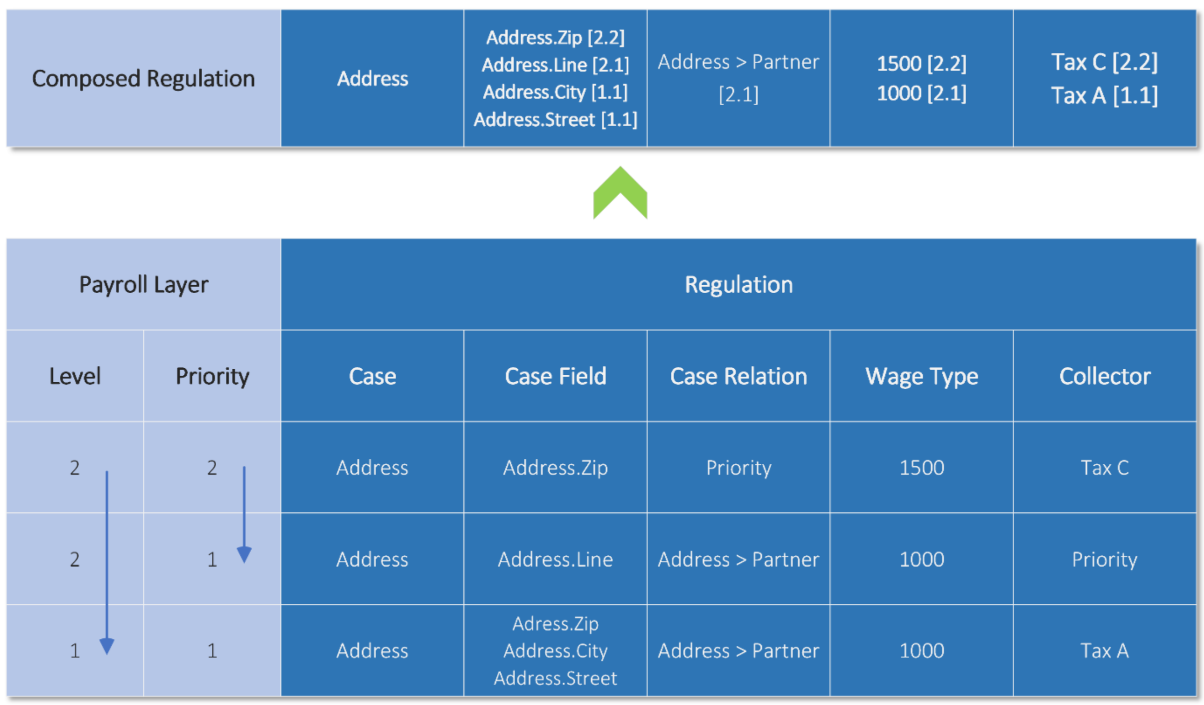
Payroll	Combines several rule sets into one payroll and is assigned to a business unit.
Payroll Layer	Determines the priorities and sequence of the regulations.

The payroll provides a consolidated view of the regulations:


Cases
Lookups
Wages
Reports
Scripts
Composed
Tenant Payroll
Tenant Regulation
Provider: Payroll Provider
Implementation Skills: low
Industry Regulations
Provider: Payroll Provider
Implementation Skills: medium
Country Regulations
Provider: Country Partner
Implementation Skills: high
Global Regulations
Provider: Payroll Provider
Implementation Skills: high

2.7.1 Payroll S Layers

The evaluation sequence of the regulations is determined in *the payroll layer* by the level (1st sort criterion) and the priority (2nd sort criterion). The number of layers is unlimited. The following example shows three Payroll Layers that are linked to form a dynamic payroll:



Composed Regulation	Address	Address.Zip [2.2] Address.Line [2.1] Address.City [1.1] Address.Street [1.1]	Address > Partner [2.1]	1500 [2.2] 1000 [2.1]	Tax C [2.2] Tax A [1.1]

Payroll Layer		Regulation				
Level	Priority	Case	Case Field	Case Relation	Wage Type	Collector
2	2	Address	Address.Zip	Priority	1500	Tax C
2	1	Address	Address.Line	Address > Partner	1000	Priority
1	1	Address	Adress.Zip Address.City Address.Street	Address > Partner	1000	Tax A

2.7.2 Object inheritance

Objects from lower Payroll Layers can be overridden by appending the overlay key. The values of all Layers are dynamically combined into one object:

- Localizations
- Attributes

- Clusters
- Case slots, lookups, and actions
- Text fields

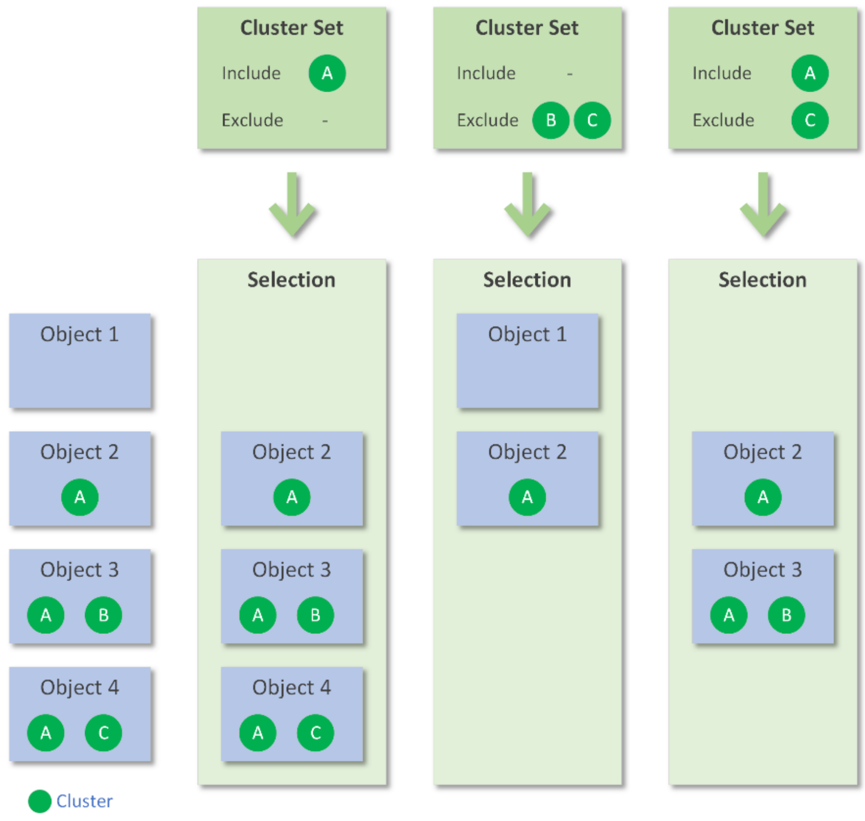
2.7.3 Payroll Cluster s

Clusters can be used to group rule objects according to any criteria. This can be used to:

- Filter API queries, e.g., to group cases
- Optimize Payruns by reducing case data
- restrict payroll results

Multiple cluster names (similar to tags) can be assigned to each cluster object. Objects are queried via the cluster set, which contains the clusters to be included (whitelist) or excluded (blacklist). By combining whitelist clusters with blacklist clusters, the cluster objects to be considered can be further restricted.

The following figure shows different scenarios for selecting cluster objects:



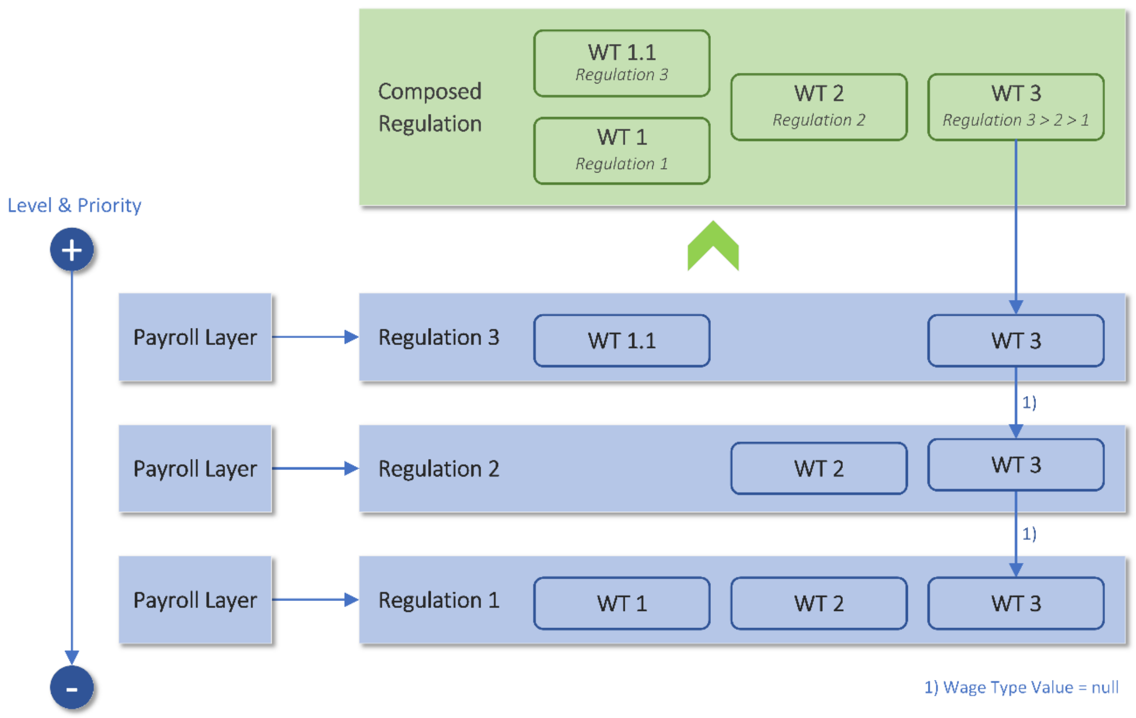
Cluster sets are defined in the *Payroll* object and can be used in a Payrun. The following cluster sets are available:

- *Case Cluster Set*: available input cases
- *Case Field Cluster Set*: available case data in the Payrun
- *Collector cluster set*: available collectors in the Payrun
- *Collector Retro Cluster Set*: available collectors in the retroactive Payrun
- *Wage Type Cluster Set*: available wage types in the Payrun
- *Wage Type Retro Cluster Set*: available wage types in retroactive Payrun
- *Wage Type Period Cluster Set*: generate period results from case data
- *Case Value Cluster Set*: Available case data in Payrun for additional results

2.7.4 Payroll Scripting

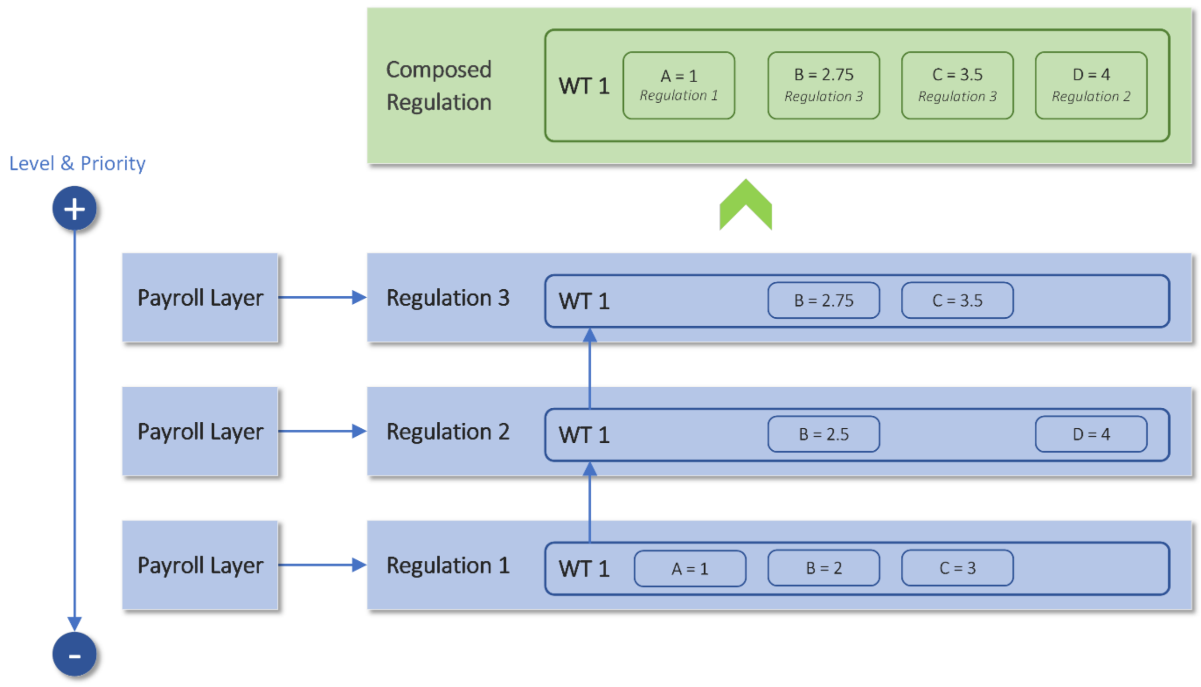
The Scripting API enables you to control the runtime behavior for specific payroll objects (seeScripting API). The scripts are evaluated using the evaluation sequence in the payroll layer (seePayroll S Layers).

The following example shows the calculation of wage types:



By returning an undefined return value (*zero*) from a scripting function, WT2 in the example above, processing is delegated to the underlying set of regulations.

In contrast, object attributes are evaluated in the opposite direction, so that the value can be overwritten by a higher-level set of Regulations:



2.8 Wage data calculation

2.8.1 Wage types

Employee wages are calculated using wage types, which are calculated in the order of their wage type number. One or more collectors can be assigned to each wage type to calculate the consecutive wage bases. The number of wage types is not limited and is determined by the sum of the payroll regulations.

2.8.2 Collectors

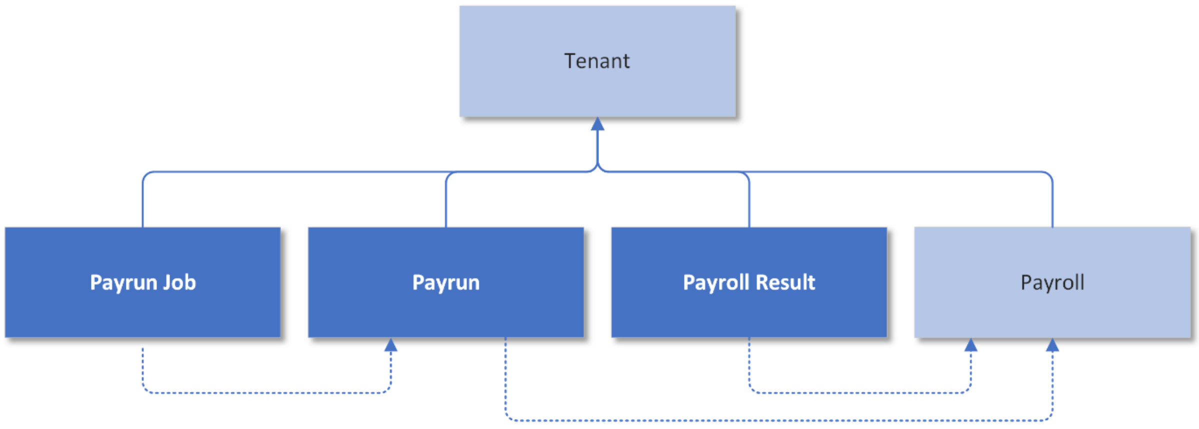
Collectors collect the values of the wage types and offer various predefined operators:

- *Sum* (default)
- *Min*
- *Max*
- *Average*
- *Count*

In addition to the predefined aggregation types, the collector value can be determined individually using the *CollectorApply* scripting function.

2.9 Payrun

Payroll data is calculated using the following objects:



Payrun Job	Execution of the Payrun.
Payrun	The definition of the Payrun.
Payroll Result	The results of the Payrun.

The Payrun sequence is described in the chapter "Scripting regulation .

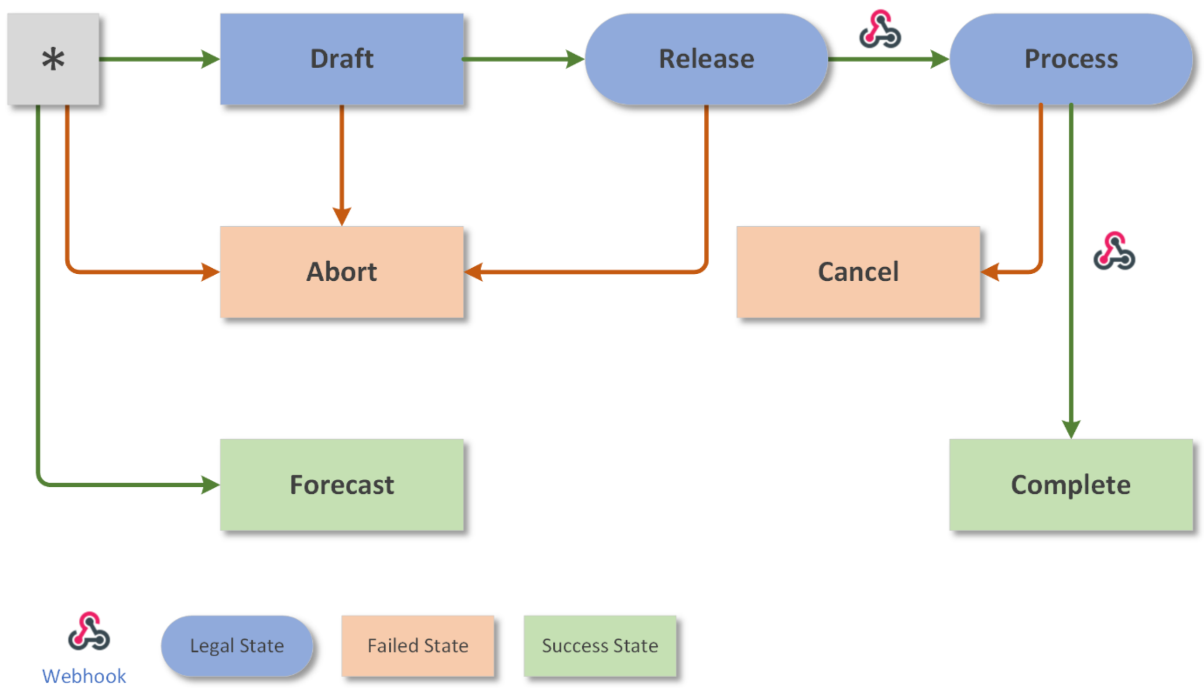
2.9.1 Payrun job

The *Payrun* job starts the Payrun for a payroll period and saves the results in the result (*Payroll Result*). The underlying payroll determines whether the employee is to be included based on the Division.

The payrun job determines the purpose for which the execution is performed:

- Statutory payroll reporting
- Forecast analysis of payroll data for forecasts, case scenarios, etc.

The Payrun job is controlled via the job status:



Job status	Type	Description	Webhook
*		New Payrun job	
Draft	Working	Legal Payrun job for preview	
Release	Working	Legal Payrun job released for processing	
Process	Working	Legal Payrun job in processing	*PayrunJobProcess*
Complete	Final	Legal Payrun job successfully processed	*PayrunJobFinish*
Forecast	Final	Payrun job	
Abort	Final	Legal Payrun job canceled before release	
Cancel	Final	Legal Payrun job processed incorrectly	*PayrunJobFinish*

For legal Payruns, there can only be one job in *Draft* status per Payrun and payroll period. In *Release* and *Process* status, multiple jobs with the same status are possible. Forecast Payruns can be executed as often as desired.

2.9.2 Payrun restart

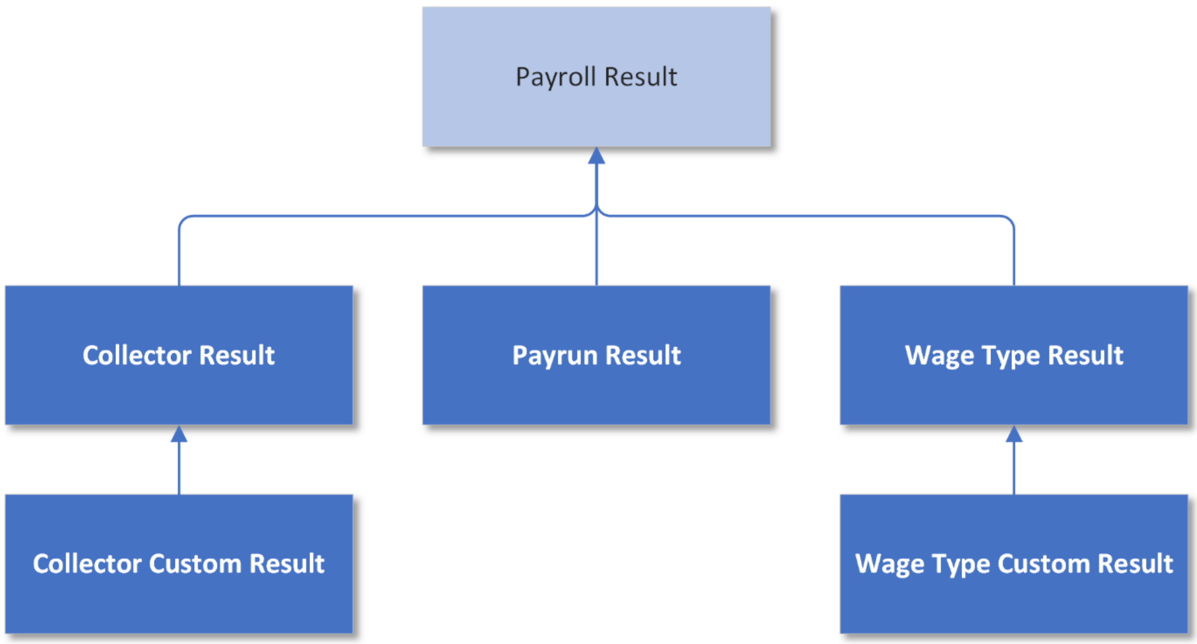
During the payrun, all wage types are processed in the order of their wage type numbers. In special cases, the payrun can be restarted for an employee. Each payrun is identified in the wage type by the run counter. Data can be exchanged between payruns using the runtime values (seePayrun Scripting).

2.9.3 Payrun control

A Payrun is started by creating Payrun jobs. While the processing of payroll data begins in the background, the API request is answered immediately.
Specific API endpoints can be used to retrieve the status of the Payrun job and the number of employees processed.
Once all employees have been successfully processed, the status changes to *Draft* or *Forecast*. In the event of an error, it changes to *Error*.

2.9.4 Payroll results

The results of the Payrun are stored in various objects:



Collector Result	The result (decimal value) of the collector.
Collector Custom Result	User-defined collector result (decimal value).
Payrun Result	Payroll-specific result (including default values).
Wage Type Result	The result of the wage type (decimal value) including user-defined attributes.
Wage Type Custom Result	User-defined wage result (decimal value).

In payroll, cluster sets (seePayroll Cluster s) are used to control which results are to be generated.

2.10 Special Payruns

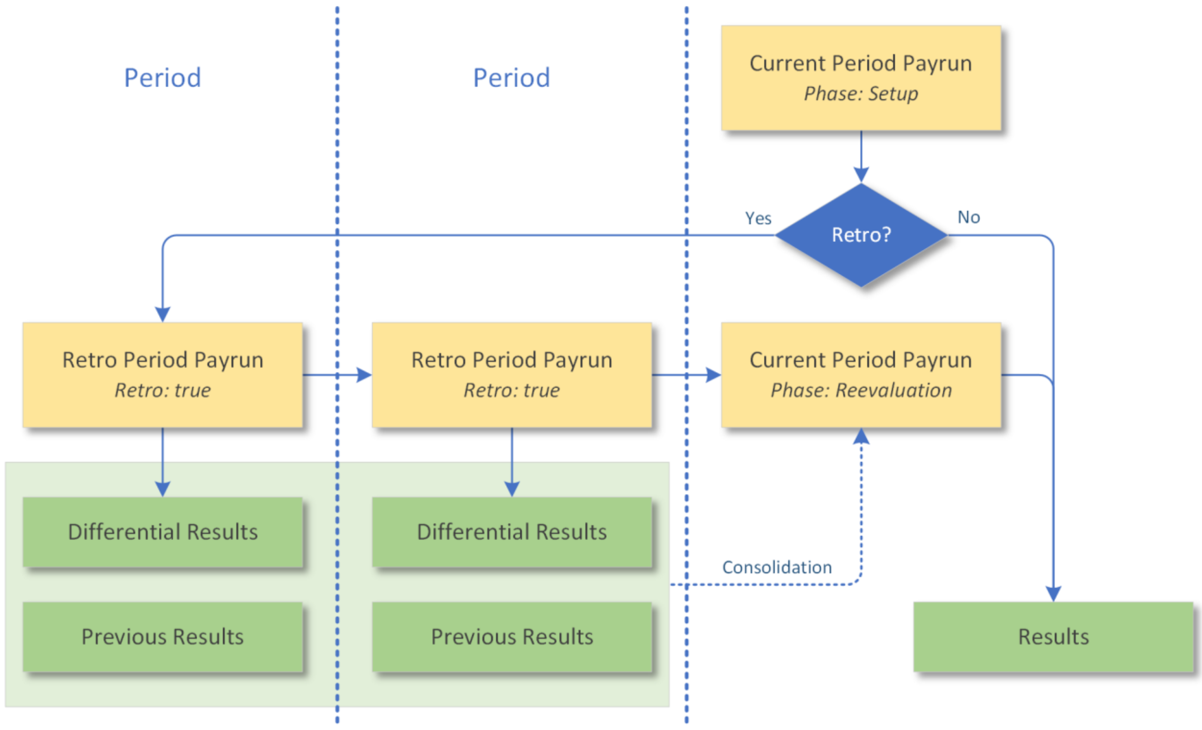
2.10.1 Incremental Payrun

If several Payruns are performed within a period, the engine only stores incremental results, i.e., values that have changed since the last Payrun. The REST API provides corresponding endpoints to determine the currently valid (consolidated) results of a payroll period.

2.10.2 Retroactive accounting

Retroactive calculations are performed automatically if new changes have occurred since the last Payrun that affect past payroll periods.

The retroactive accounting process:



The steps of the retroactive accounting process are:

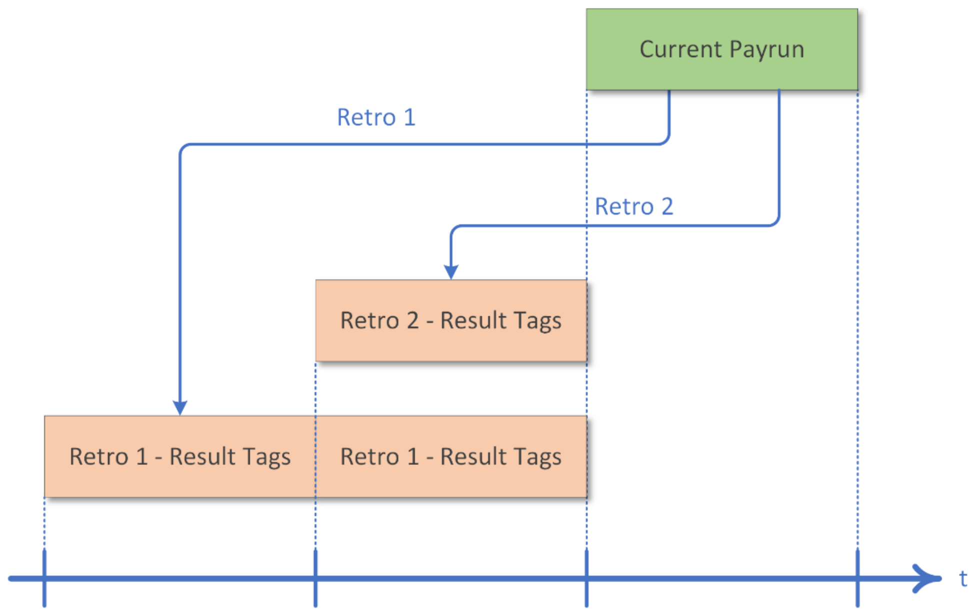
1. Calculation of the current payroll period (with change to the previous period)
   → Execution phase *setup*
   → Transient results
2. Calculation of all previous periods starting with the change period up to the previous period
   → Save incremental results of previous periods
3. Recalculation of the current payroll period
   → Execution phase *revaluation*
   → Access consolidation results taking into account retro results
   → Save results

For forecasts, the runs with the same forecast name apply. The number of periods in the retroactive calculation is unlimited, but it is possible to limit the retroactive calculation to the payroll cycle.

2.10.3 Manual retroactive accounting

In addition to automatic retroactive accounting, retroactive accounting can also be triggered manually using scripts. The results generated during retroactive accounting are marked using payroll result tags. The tags can be used as filter criteria when querying the payroll results.

In the following scenario, the Payrun for the current period triggers two retroactive accounting Payruns:



2.11 Forecast

Two actions are required to create forecasts:

- Changing the case data for a forecast
- Executing a Payrun job for a forecast

When the payrun for a forecast is started, all case data for the forecast is available, as well as all other case data that is not assigned to a forecast. The normal payrun, on the other hand, ignores the case data from forecasts.

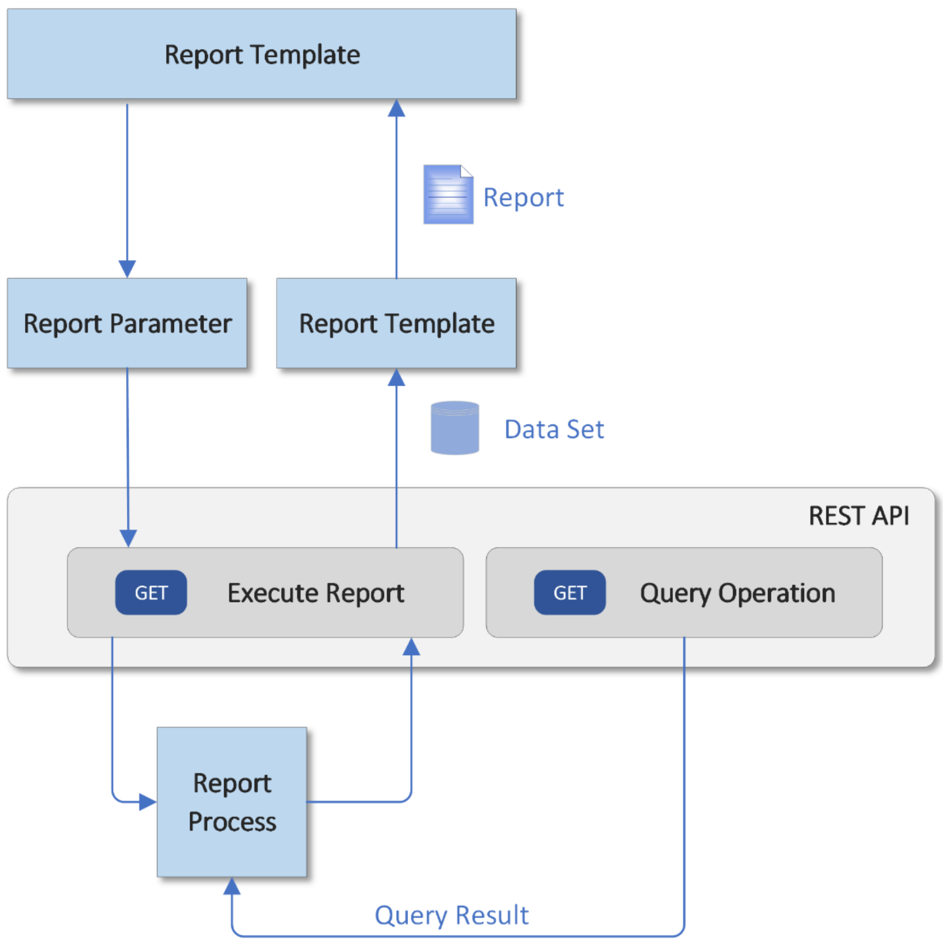
2.12 Report

Report generation takes place in three steps, each of which is optional:

1. Report preparation: *Report build*
2. Report queries: *Report start*
3. Post-processing of query data: *Report End*

The Payroll API calculates the report data in a dataset (tables and relations) that is transformed by the client into the target document. All GET endpoints of the REST API serve as data sources for determining the report data.

Report generation:

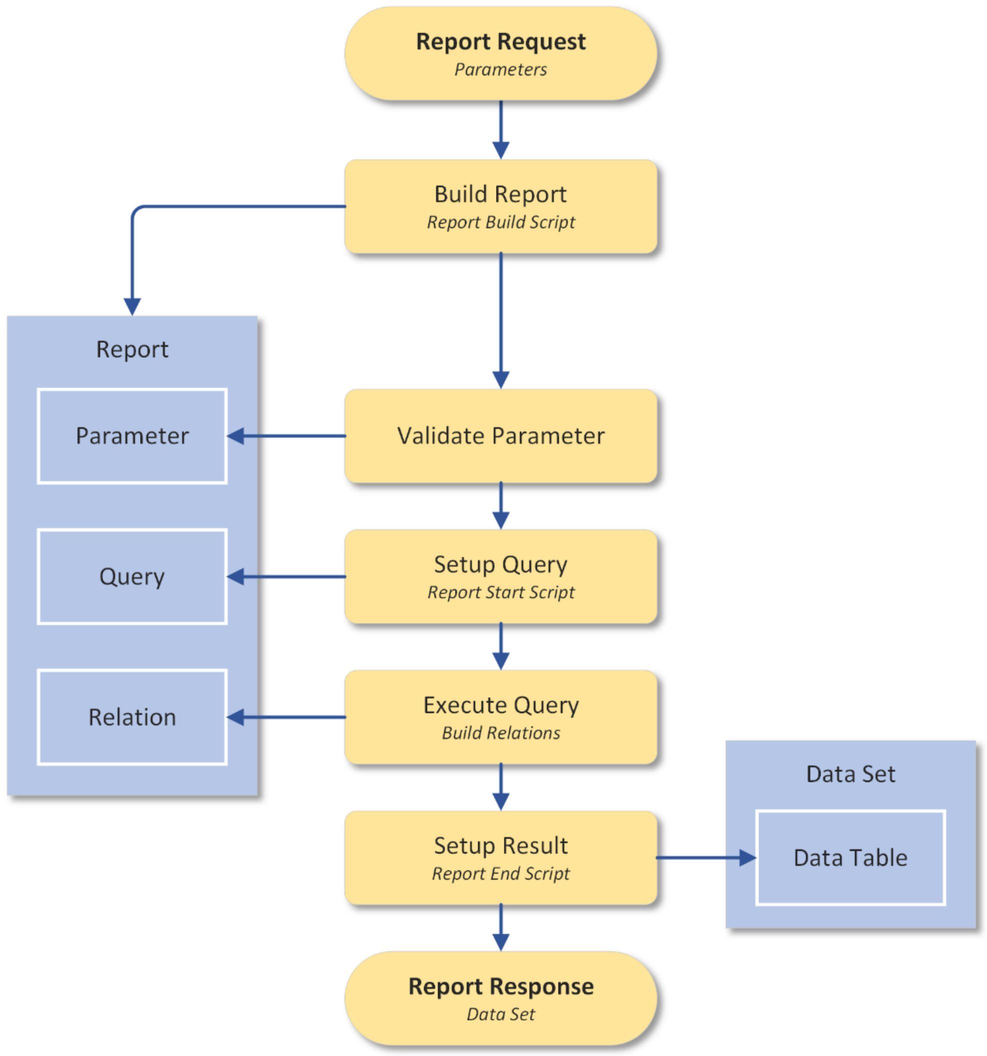


The language-dependent report layout (*ReportTemplate*) is irrelevant to the calculation of the report data and is determined by the REST client (*.frx, .docx, .rdlc, etc.)*.

The report layout also contains data fields for schemas that the REST client can use to validate the generated report data (JSON/XML/etc.).

2.12.1 Report processing

A report is created in the following steps:



1. Report creation: The report parameters can be customized in *the Report BuildScript*.
2. Check and complete the report parameters.
3. Prepare the queries: The queries are predefined in the report and can be customized in the *Report Start Script*.
4. Query the report data: The report can execute multiple queries, whereby the data from an API endpoint (GET) is stored in a table (memory). Finally, the relationships between the tables are established.
5. Further processing of the report data: In *the Report End Script*, the tables can be adjusted to the desired target format. Runtime queries are possible in the script to handle complex cases.

Each step is optional and can vary depending on the type of report. A simple report can consist of queries only. A complex report can contain the entire logic of the post-processing logic.

Additional report functions:

- Access to localized texts
- Output of user-defined attributes
- Updating of document metadata, including query parameters
- Extensive editing options for report tables via scripting
  (Microsoft https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset)
- Categorization of reports to restrict selection in the REST client with clusters

2.12.2 Report logs

The execution of a report can be logged in the log and serves as a history for reports that require information from previous reports, such as the reporting date or reporting status.

3 Basic concepts

3.1 Languages

The Payroll service supports all RFC 4646 languages with English as the translation basis. The English identifier is used for references between objects.

3.2 Audit Trail

The engine logs every change to the central payroll data in the audit trail. This applies to all rule objects and case data. The audit trail for case data also serves as the basis for calculating period values.

3.3 Data Types

The engine offers the following data types:

Data type	JSON	C	Domain
String	String	String	
WebResource	String	String	RFC 1738
Date, DateTime	String	DateTime	ISO 8601
Integer	Integer	Int32	Signed 32-bit integer
Decimal, Money, Percent, Hour, Day, Week, Month, Year, Distance, NumericBoolean	Number	Decimal	28 decimal places 6 decimal places
Boolean	Boolean	Boolean	
None	Zero	zero	

3.4 Calendar

The calendar divides the data into two time units:

- **Period** in which wages are paid, usually the calendar month
- **Cycle** in which the wage periods are repeated, usually the calendar year

The wage cycle is always a multiple of the wage period. The following overview shows the possible combinations of the two time units with the number of wage payments within a cycle:

Cycle → / Period ↓	Week	Two weeks	Half-month	Moon Month	Month	Two months	Quarter	Half-year	Year
Week	1	2	-	4	-	-	13	26	52
Two weeks	-	1	-	-	-	-	-	13	26
Half-month	-	-	1	-	2	4	6	12	24
Moon Month	-	-	-	1	-	-	-	-	13
Month	-	-	-	-	1	2	3	6	**12**
Two months	-	-	-	-	-	1	-	3	6
Quarter	-	-	-	-	-	-	1	2	4
Half-year	-	-	-	-	-	-	-	1	2
Year	-	-	-	-	-	-	-	-	1

3.4.1 Calendar configuration

The engine's time calculations are based on the following calendar configuration:

- Time unit of the cycle
- Time unit of the period
- Time value cyclical or periodic, examples:
  - Cycle=year, period=month, base value=60,000→ wage value=5,000
  - Cycle=half-year, period=two weeks, base value=52,000→ wage value=4,000
- First month of the year for fiscal years (default: January)
- Rule for the first week of the year
- First day of the week (Monday)
- Average number of days in the month with an average value
- Definition of days in the working week (e.g., Monday to Friday)

The calendar is determined by the following objects and can be overridden at various levels. The following calendar users are available in the evaluation priority:

1. Wage type (temporary in the Payrun)
2. Employee
3. Division
4. Tenant

3.4.2 Universal time

The date values of the REST API must be specified in UTC; other time models will result in errors.

3.4.3 Time types

The calculation of time values is based on the following time types:

Type	Description	Periods Start	Period end	Number Period values
Timeless	Value without time constraint e.g., duty of care	-	-	1
Moment	Value assigned to a point in time e.g., bonus	x	-	1..n Sum

Period	Value for a period e.g., working hours	x	(x)	n Overlaying
CalendarPeriod	Value distributed on a daily basis over the calendar period e.g. monthly salary	x	(x)	n overlapping

3.5 Object properties

Each API object has the following basic properties:

- Identification, sequential number within its type
- Engine time of creation in UTC (*Created*)
- Engine time of last change in UTC (*Updated*)
- Object status, active (default) or inactive

3.6 Object attributes

Object attributes are user-defined data whose content has no influence on the behavior of the engine. Attributes can be used for the following scenarios:

- References to external objects in surrounding systems
- Storage of additional data (if no time value or case value)
- Control in client applications, e.g., control of case data entry
- Control/parameterization of scripting functions (collector or wage type)
- Storage of additional wage type results with, for example, notification data

The attributes are managed in a value list (dictionary) in the object. Some objects offer API endpoints for editing the object attributes directly.

Attribute fields can be used for filtering in REST queries (seeAttribute queries). Overview of API objects with attributes:

Attribute object	**Attribute API**	**Audit**
Tenant	x	
User	x	
Calendar	x	
Task	x	
Division	x	
Employee	x	
Payroll	x	
Payroll Layer	x	
Payrun Job	x	
Webhook	x	
Regulation	x	
Case		x
Case Field		x
Case Relation		x
Collector		x
Wage Type		x
Report	x	x
Case Value		
PayrollResult		

4 Payroll Tests

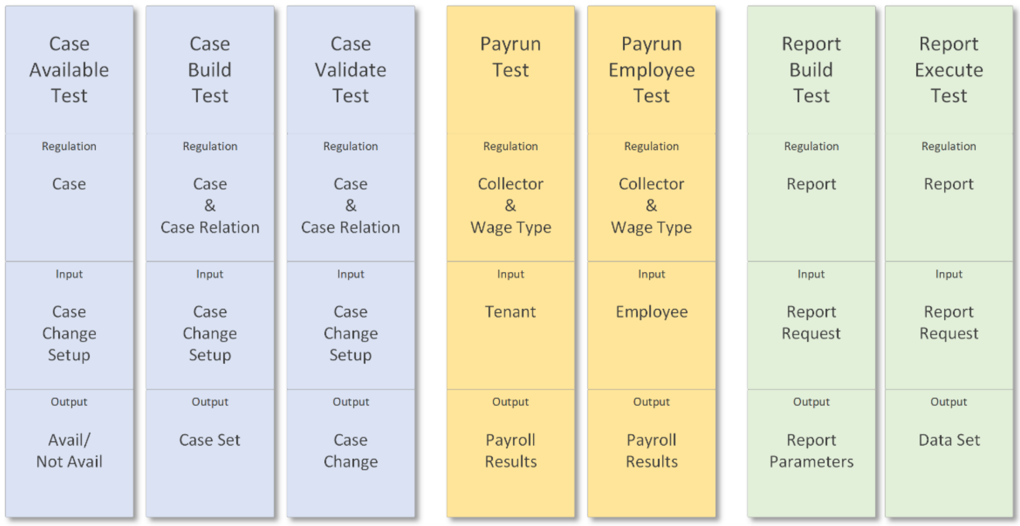
Based on modern approaches to software development, the Payroll Engine also offers test-driven development of payroll calculations. With the help of Client Services (seeClient Services), the following aspects of payroll accounting can be tested automatically:

- Employee cases in the past and future
- Any payruns within a payroll period and across multiple payroll periods
- Comparison of calculated payroll data with expected results

Automated testing is possible in three areas:

- Testing a case
- Testing a payrun for a tenant or an employee
- Testing a report

All objects in the regulations can be tested automatically using the following tests:



	Case Available Test	Case Build Test	Case Validate Test	Payrun Test	Payrun Employee Test	Report Build Test	Report Execute Test
Regulation	Case	Case & Case Relation	Case & Case Relation	Collector & Wage Type	Collector & Wage Type	Report	Report
Input	Case Change Setup	Case Change Setup	Case Change Setup	Tenant	Employee	Report Request	Report Request
Output	Avail/ Not Avail	Case Set	Case Change	Payroll Results	Payroll Results	Report Parameters	Data Set

4.1 Case tests

Case tests can be used to test the availability, generation, and validation of a business case. The test cases can be defined by configuration (JSON) or by programming (C# with Client Services).

4.2 Payrun tests

4.2.1 Tenant Payrun tests

In the client Payrun test, a new tenant is created and the results of the Payruns are compared with the expected results. After the test, the tenant is deleted.

4.2.2 Employee Payrun tests

For more extensive tenant scenarios, tests for individual employees are possible. The employee payrun test requires an existing tenant. The payrun is performed on a copy of an existing employee. The test data is cleaned up manually.

4.3 Report tests

For payroll reports, both report creation (report parameters) and report execution can be tested. Report tests can be defined through configuration (JSON) or programming (C# with Client Services).

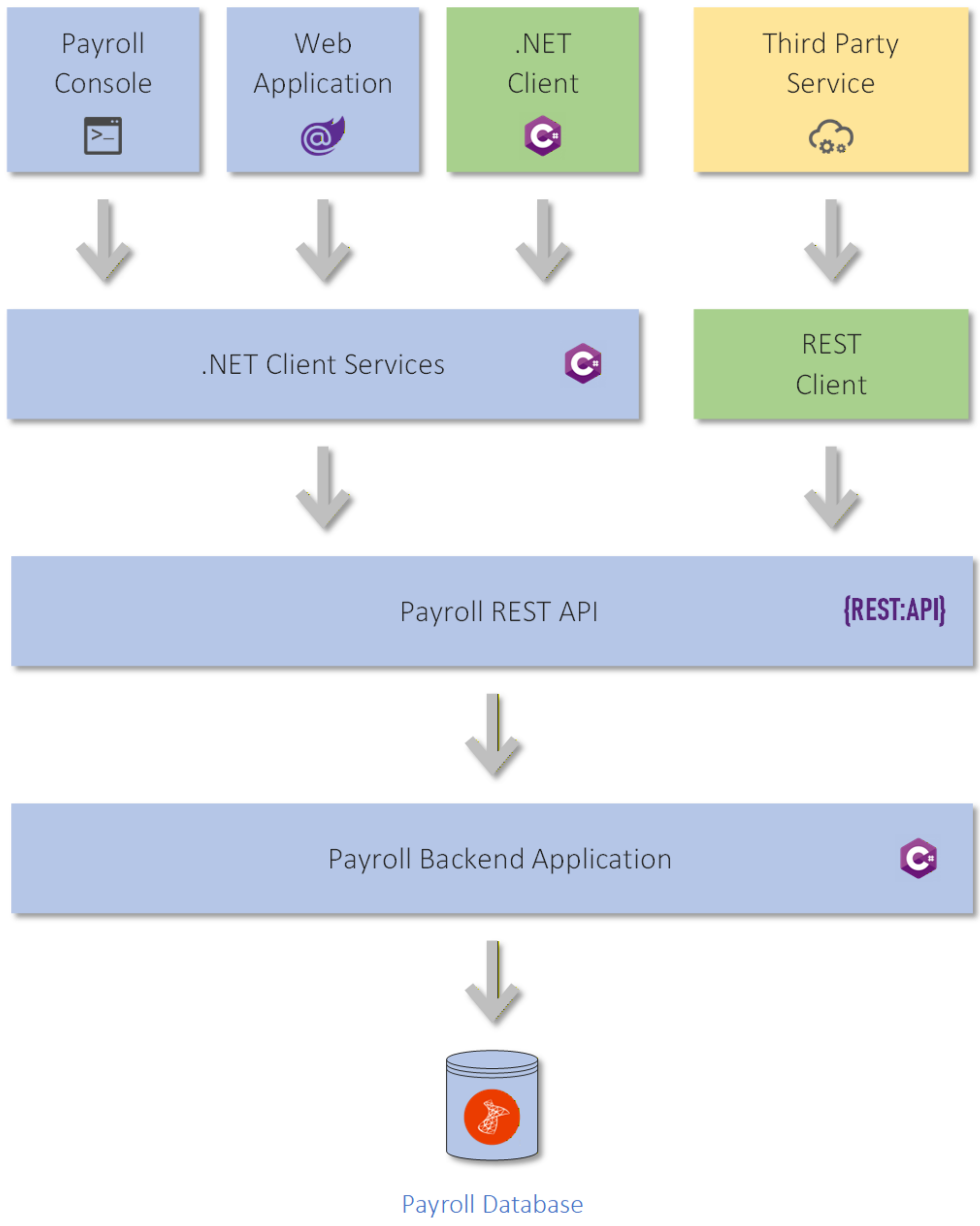
4.4 Applying tests

The test is defined in JSON and executed using the Payroll Console application. The test results are displayed on the screen and test errors are logged in log files.

The test functions are also available as programming modules (seeClient Services) and can be individually extended.

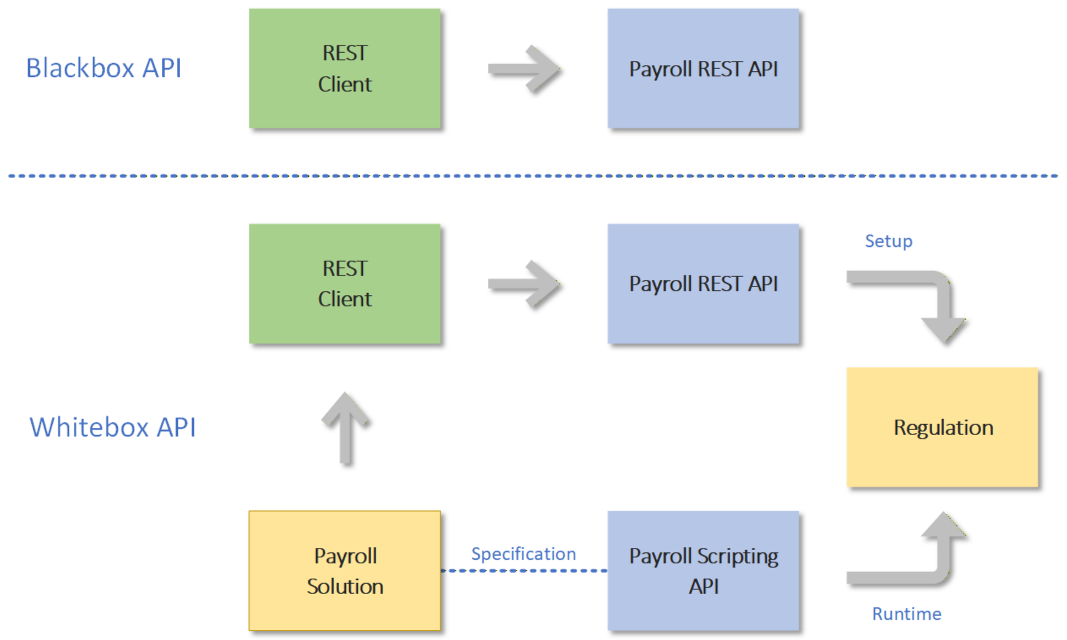
5 Payroll Backend

The payroll backend consists of the following components:



5.1 Whitebox API

The Payroll Engine is a white box REST API with an additional interface, the Scripting API, which can be used to influence runtime behavior. The following figure shows the differences between the black box and the white box REST API:



5.2 Technologies

The following technologies are used in the Payroll Backend:

•	REST API	OpenAPI 3.1 (OAS3.1)
	Media type	application/json
	Description	*swagger.json* (see GitHub Release Asset)
•	Scripting API	C
•	Applications	.NET 10
•	Database	SQL Server 2017 or newer
•	Development packages	NuGet packages
•	Application packages	Zip download (see GitHub Release Asset) or Docker container for backend server, payroll console, and web app

5.3 List queries

The following criteria can be used to restrict data in API queries:

- Status by active or inactive objects
- Filter expression by OData
- Field order according to OData
- Field selection according to OData
- Page restriction with the number of objects per page and the page offset

5.3.1 Attribute queries

The engine offers the option to filter, sort, and group list queries by attribute values. The prefix in the attribute name determines the data type of the attribute:

- Text attribute *TA_*
  → Example: *TA_MyName*
- Date attribute *DA_*
  → Example: *DA_ProjectCreated*
- Number attribute *NA_*
  → Example: *NA_SecurityLimit*

5.4 Webhooks

The engine offers webhooks for the following actions:

Action	Trigger	Webhook message	Track
CaseFunctionRequest	*Case* Function Request	*Operation: <Custom>* *Request: <Custom>*	-
CaseValueAdded	New Case Mutation	*Request: CaseChange*	✓
PayrunFunctionRequest	*Payrun* function request	*Request: <Custom>* *Operation: Function name*	-
PayrunJobStarted	Legal Payrun job released for processing	*Request: PayrunJob*	✓
PayrunJobCompleted	Legal Payrun job completed	*Request: PayrunJob*	✓
ReportFunctionRequest	*Report* function request	*Operation: <Custom>* *Request: <Custom>*	-
TaskChange	Task Change	*Request: TaskChange*	

The webhook method is called using HTTP POST and receives a *webhook message* as an argument. The HTTP status code and content (JSON) are evaluated in the backend from the webhook response. Specific webhook messages are tracked in the backend and can be queried via the REST API.

5.5 Administration API

The Payroll API offers various endpoints for system administration:

- Terminating the application
- Clearing the application cache
- Determining the available report endpoints (seeReport)

5.6 Application log s

For system analysis, the engine logs certain events in log files that do not contain any sensitive information. The log files are stored in the Windows system directory for program data (*%ProgramData%*).

In addition to the application log, there are also customer-specific logs:

- Log for payruns (seePayrun logs)
- Log for reports (seeReport logs)

5.7 API security

The Payroll Engine API offers the following security mechanisms:

5.7.1 API key

A fixed key can be stored in the backend server configuration, which must be included in the request header for each endpoint. This mechanism prevents unprotected access to the API.

5.7.2 Tenant authentication

To prevent unauthorized access between tenants, the backend server supports the HTTP header *Auth-Tenant*. If this optional header is present, it is compared with the tenant of the URL for each REST request.

6 Scripting API

The Scripting API allows the user to influence the runtime behavior of the engine:

Case Management

- Determine the availability of a case
- Set up a case and determine case values
- Define behavior between cases
- Case validation

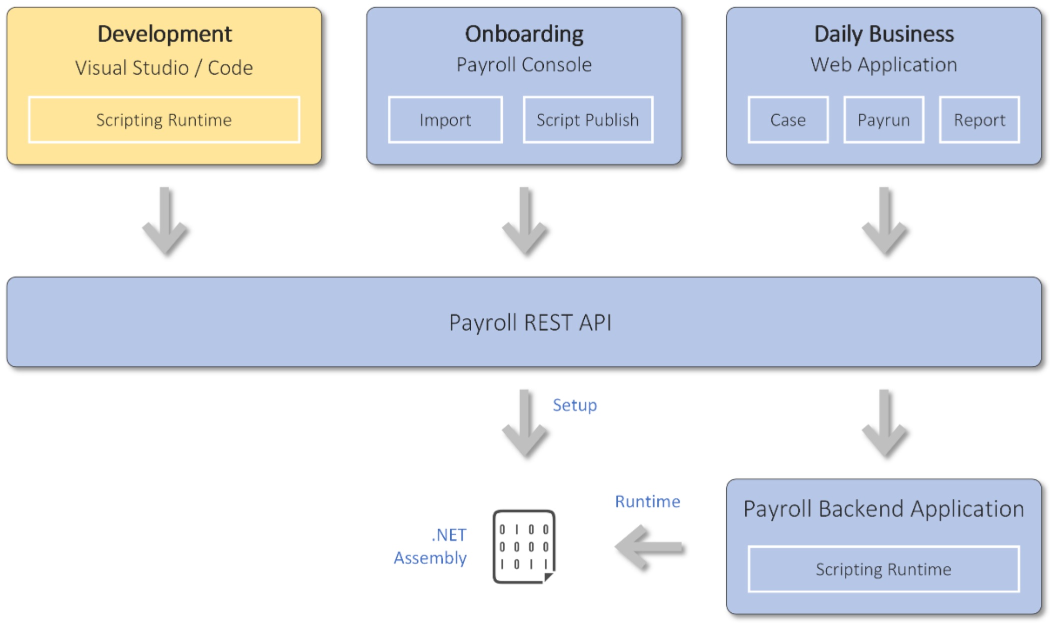
Payrun

- Restrict employees and wage types
- Calculate and save wages with collectors and wage types
- Communication with external services (webhooks)
- Calculate and save additional payroll data

Report

- Define report parameters
- Calculate report data

The following figure shows the relationship between the Payroll REST API and the Scripting API:

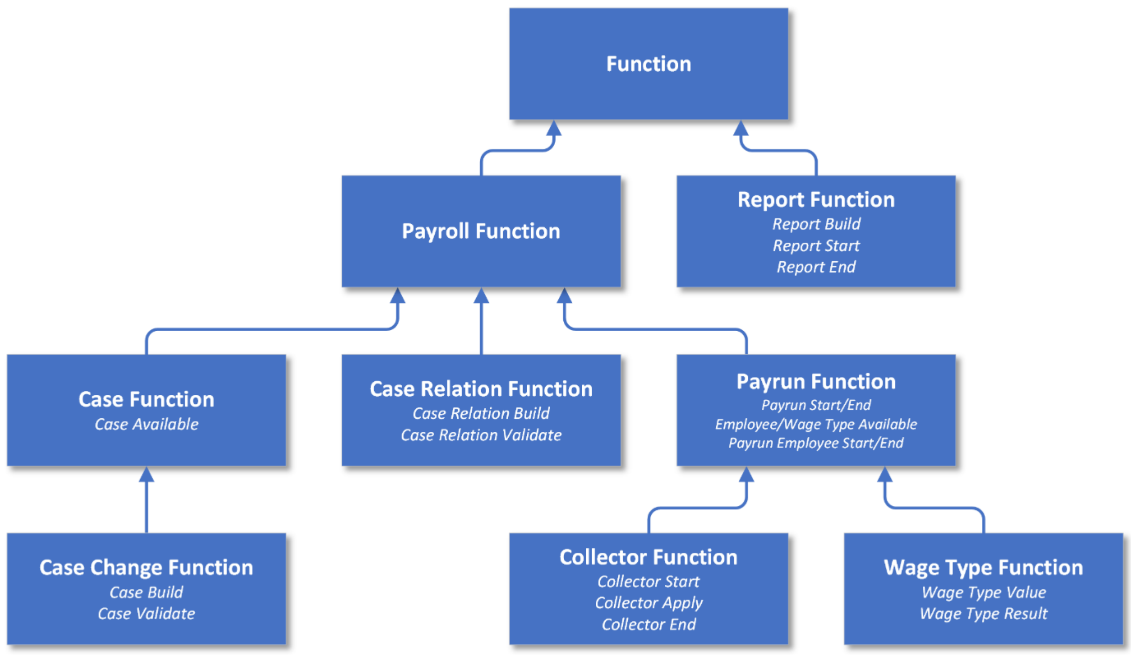


During onboarding, scripts are developed and transferred to the REST API using the *Payroll Console*. When the scripts are imported, they are translated into machine language (compiled). At runtime, the binary scripts are loaded and executed by the payroll process. The script code can use the functions of the Scripting API.

During scripting development, scripts can be tested locally for debugging purposes.

6.1 Functions

A function is a programmed sequence that is called by an engine process at specific points. The functions are divided into the following areas:



The available scripting functions are:

Object	Function	Description	Type	Overlay
Case	Available	Test whether a case is available	bool?	1. Top-down
Case	Build	Create case	bool?	1. Top-down
Case	Validate	Test whether a case is valid	bool?	1. Top-down
Case relation	Build	Create case relationship	bool?	1. Top-down
Case Relation	Validate	Test whether a case relationship is valid	bool?	1. Top-down
Payrun	Start	Payrun start	bool?	-
Payrun	EmployeeAvailable	Test whether the employee is available	bool?	-
Payrun	EmployeeStart	Payrun Employee Start	bool?	-
Payrun	WageTypeAvailable	Test whether a wage type is available	bool?	-
Payrun	Employee End	Payrun Employee End	-	-
Payrun	End	Payrun end	-	-
Collector	Start	Collector start	-	1. Top-down
Collector	Apply	Wage type Apply value	decimal?	1. Top-down
Collector	End	Collector end	-	1. Top-down
Wage Type	Value	Calculate wage type value	decimal?	1. Top-down
Wage Type	Result	Determine wage type results	-	* Bottom-Up
Report	Build	Prepare report	bool?	-
Report	Start	Start of report generation	-	-
Report	End	End of report creation	-	-

The "1. Top-Down" overlay ensures that the first function that returns a value is used by the billing regulations. If an undefined value is returned, the function of the underlying regulation is executed.

With the "* Bottom-Up" overlay, the data is merged starting with the lowest layer of regulations, and it is possible to overwrite values in the higher-level regulation.

Both scenarios are described in the chapter "Payroll Script .

6.1.1 Extending functions

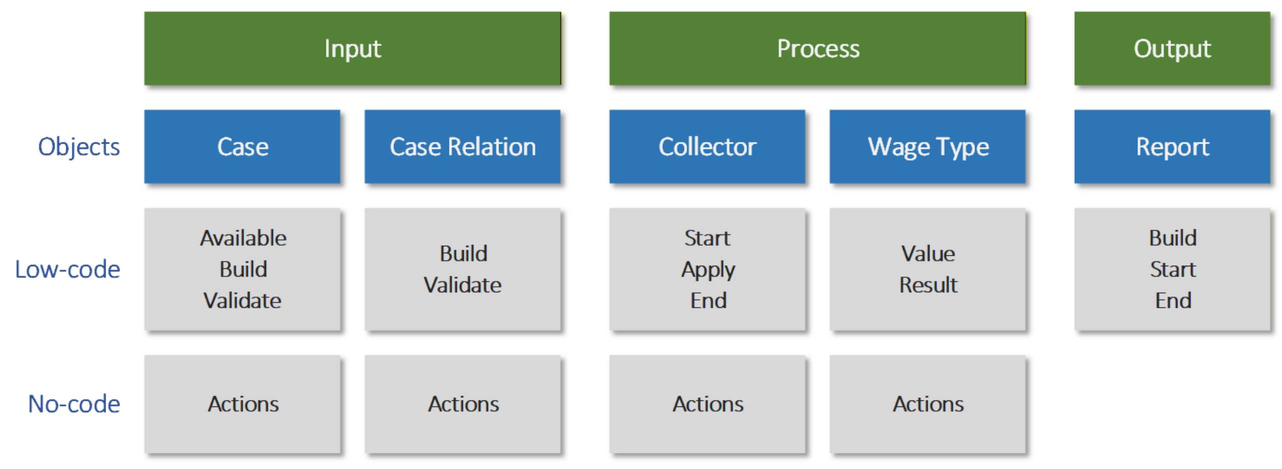
The integrated functions can be extended using scripts. There are two options for this in C#:

- Extension methods (Microsoft Extension Methods)
- Partial classes (Microsoft Partial Classes)

Partial classes are suitable for outsourcing business code to a separate class model. In case scripting, extension methods offer the option of defining the creation and validation for each case field.

6.2 Scripting regulation

To control runtime behavior, Payroll offers scripting functions for rule set objects that are described using C# low-code. With the exception of reports, the functions can be controlled via no-code actions:

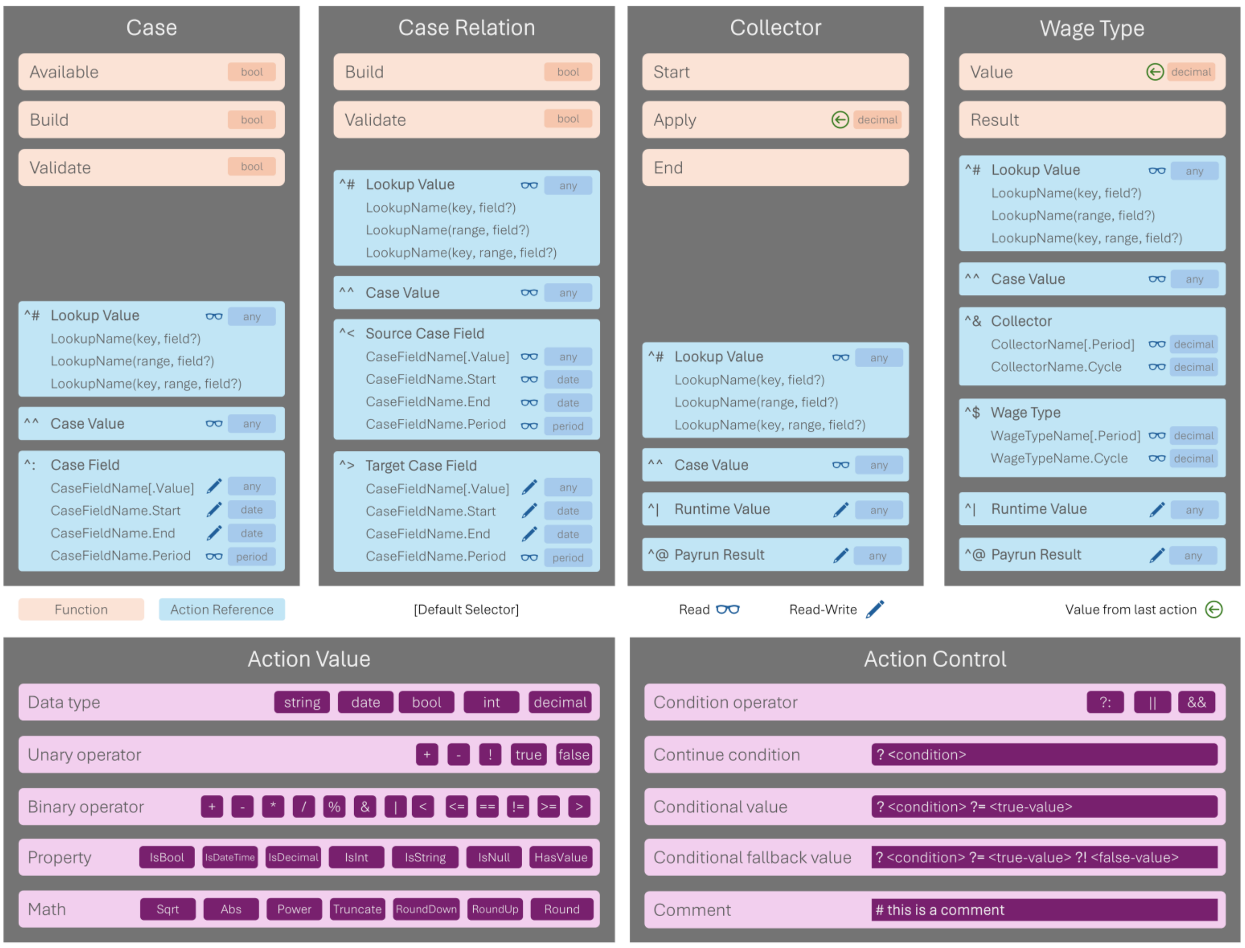


6.3 No-code

Actions can be used to define the behavior of rule objects without programming knowledge.

- Access to values from lookup tables
- Access to employee and company data
- Input
  - Control of input fields
  - Checking and setting case values
- Process
  - Calculation of collectors
  - Calculation of wage types

The following overview shows the command set of Actions:

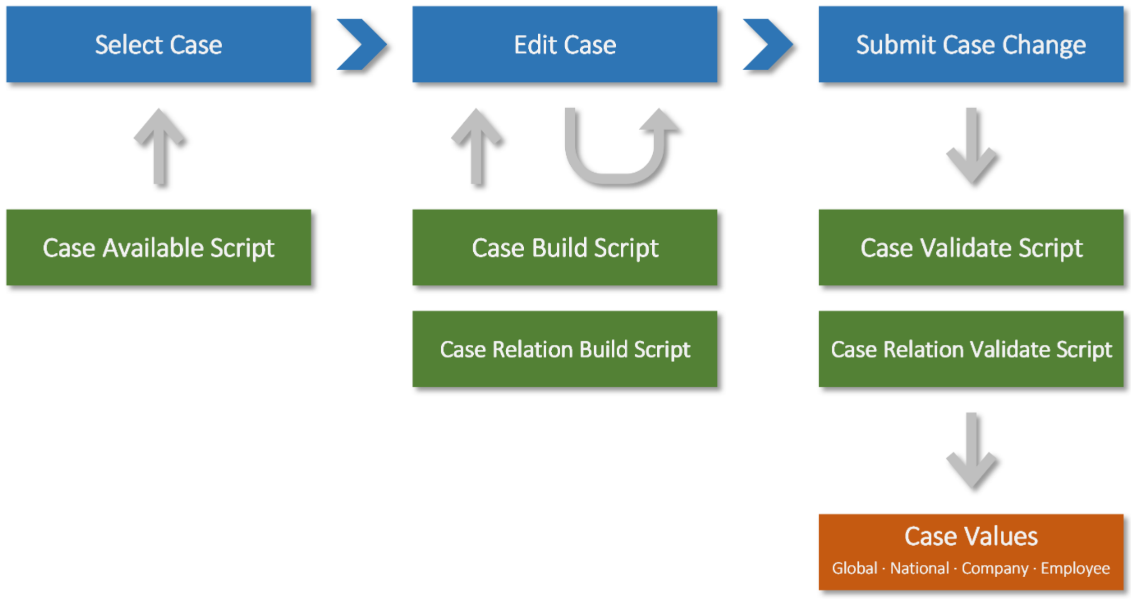


Case
Available bool
Build bool
Validate bool
^# Lookup Value any
LookupName(key, field?)
LookupName(range, field?)
LookupName(key, range, field?)
^^ Case Value any
^: Case Field
CaseFieldName[.Value] any
CaseFieldName.Start date
CaseFieldName.End date
CaseFieldName.Period period
Case Relation
Build bool
Validate bool
^# Lookup Value any
LookupName(key, field?)
LookupName(range, field?)
LookupName(key, range, field?)
^^ Case Value any
^< Source Case Field
CaseFieldName[.Value] any
CaseFieldName.Start date
CaseFieldName.End date
CaseFieldName.Period period
^> Target Case Field
CaseFieldName[.Value] any
CaseFieldName.Start date
CaseFieldName.End date
CaseFieldName.Period period
Collector
Start
Apply decimal
End
^# Lookup Value any
LookupName(key, field?)
LookupName(range, field?)
LookupName(key, range, field?)
^^ Case Value any
^| Runtime Value any
^@ Payrun Result any
Wage Type
Value decimal
Result
^# Lookup Value any
LookupName(key, field?)
LookupName(range, field?)
LookupName(key, range, field?)
^^ Case Value any
^& Collector
CollectorName[.Period] decimal
CollectorName.Cycle decimal
^$ Wage Type
WageTypeName[.Period] decimal
WageTypeName.Cycle decimal
^| Runtime Value any
^@ Payrun Result any
Function
Action Reference
[Default Selector]
Read
Read-Write
Value from last action
Action Value
Data type string date bool int decimal
Unary operator + - ! true false
Binary operator + - * / % & | < <= == != >= >
Property IsBool IsDateTime IsDecimal IsInt IsString IsNull HasValue
Math Sqrt Abs Power Truncate RoundDown RoundUp Round
Action Control
Condition operator ?: || &&
Continue condition ? <condition>
Conditional value ? <condition> ?= <true-value>
Conditional fallback value ? <condition> ?= <true-value> ?! <false-value>
Comment # this is a comment

In addition to the predefined actions, low-code can also be used to define user-defined actions in the regulation.

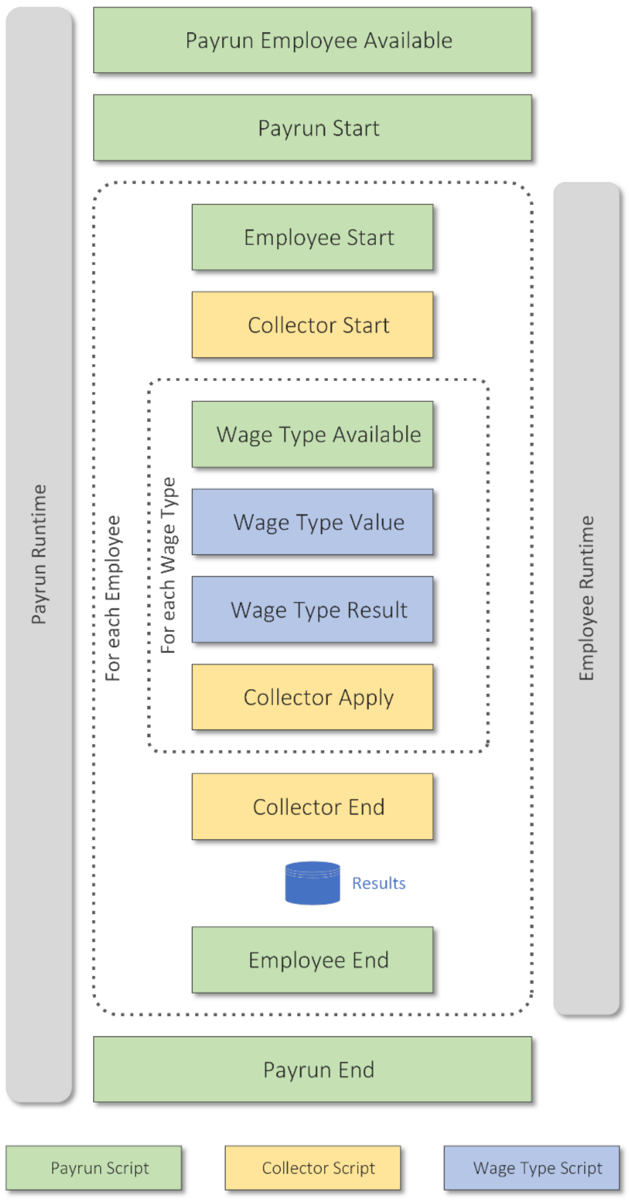
6.4 Case scripting

The lifecycle of a payroll case consists of the phases availability, generation, and transmission. In the generation phase, the case is created for the first time and rebuilt each time the data changes. This results in the following processing sequence:



6.5 Payrun Scripting

The following scripting functions control the Payrun:



During the Payrun, values can be exchanged between the functions. A distinction is made between Payrun runtime values (*PayrunRuntime*) and employee runtime values (*EmployeeRuntime*). The Payrun runtime values exist for the entire Payrun, while the employee runtime values exist for each employee. At the end of the Payrun, all runtime values are available for further processing.

Overview of Payrun runtime values:

Function	Payrun runtime values	Employee runtime values
Payrun	Access	-
Collector	Access (inherited)	Current employee
Wage Type	Access (inherited)	Current employee
Payrun End	Access (inherited)	All employees

If the wage type is assigned to one or more collectors, its value (*Wage Type Value/Result*) is applied to the collectors (*CollectorApply*).

6.6 Payrun time value calculation

When calculating a period value, all relevant changes within the period are divided proportionally according to the calendar (e.g., salary adjustments during the month). The Scripting API offers the option of using the distribution logic of the calendar, which greatly simplifies the calculation syntax. Two case values with different mutations can be calculated using the usual mathematical operators (addition, subtraction, multiplication, and division).

6.7 Payrun logs

Log entries can be generated at various levels using scripting functions. The logs are assigned to the Tenants and can be evaluated via the API. A start parameter for the Payrun job can be used to specify which log levels are to be logged.

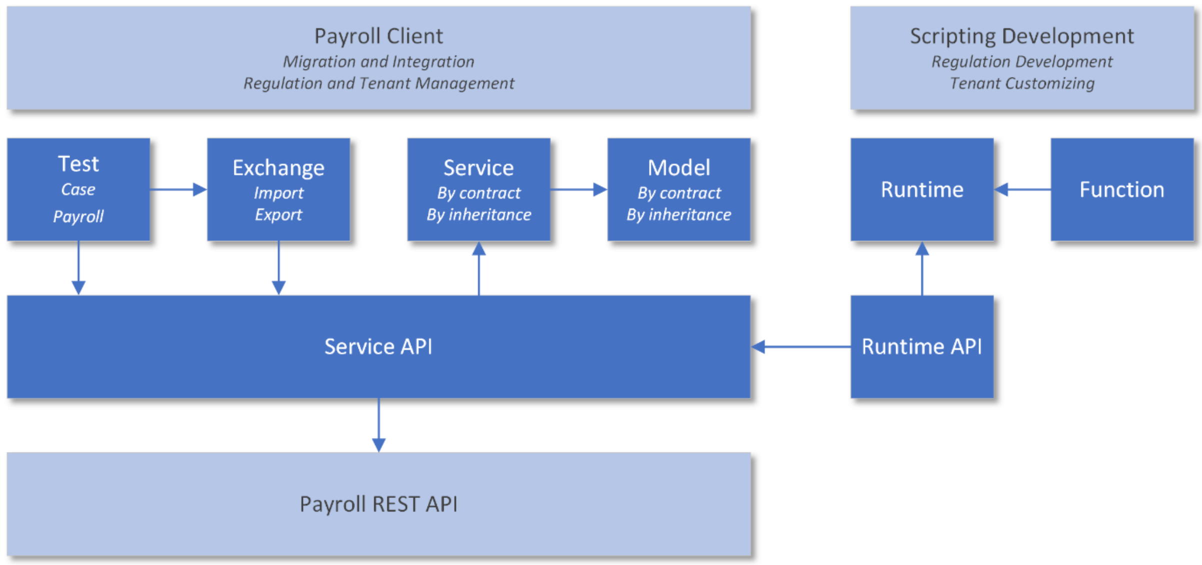
6.8 REST API scripts

Scripts are managed as object values (e.g., wage type *ValueExpression* and *ResultExpression*) and are compiled into machine code when the object is read (POST). In the event of a syntax error, the payroll service returns the error code "*422 Unprocessable Entity*."

7 Client Services

Client services are additional services that enable more efficient use of the Payroll API for

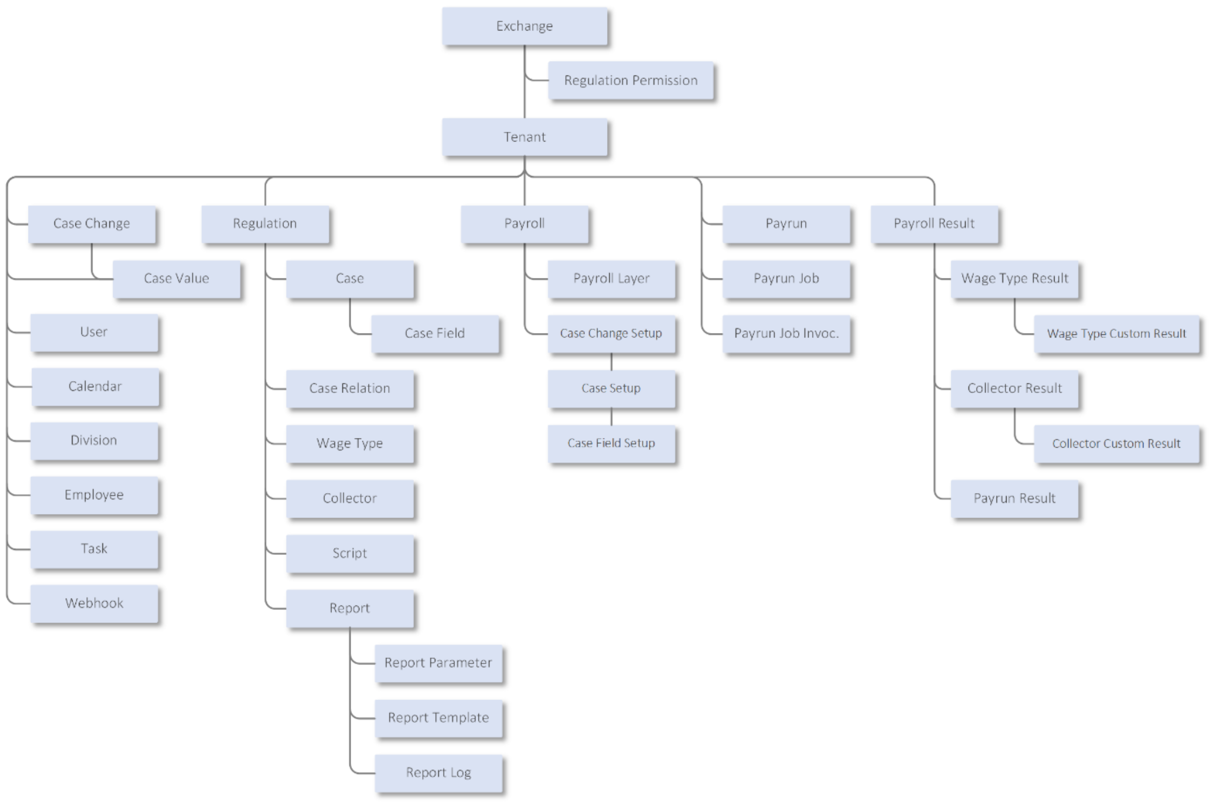
- migrations and integrations (e.g., interfaces)
- for managing regulations and tenants
- testing Regulations
- developing scripts



Model	Model of API objects (Swagger schema) extended with exchange-relevant values and properties. A model object is copyable/comparable and can be used or extended through inheritance or contract/interface.
Service	Access to the API endpoints (Swagger endpoints). A service can be used or extended through inheritance or contract/interface.
Exchange	JSON import and export of payrolls.
Test	Testing of business cases and Payruns.
Service API	Communication with the payroll API.
Function	Templates for developing function scripts.
Runtime	Function runtime environment for developing function scripts.
Runtime API	Communication with the service API.

7.1.1 Exchange model

The exchange model contains all API objects of the payroll engine for data exchange:

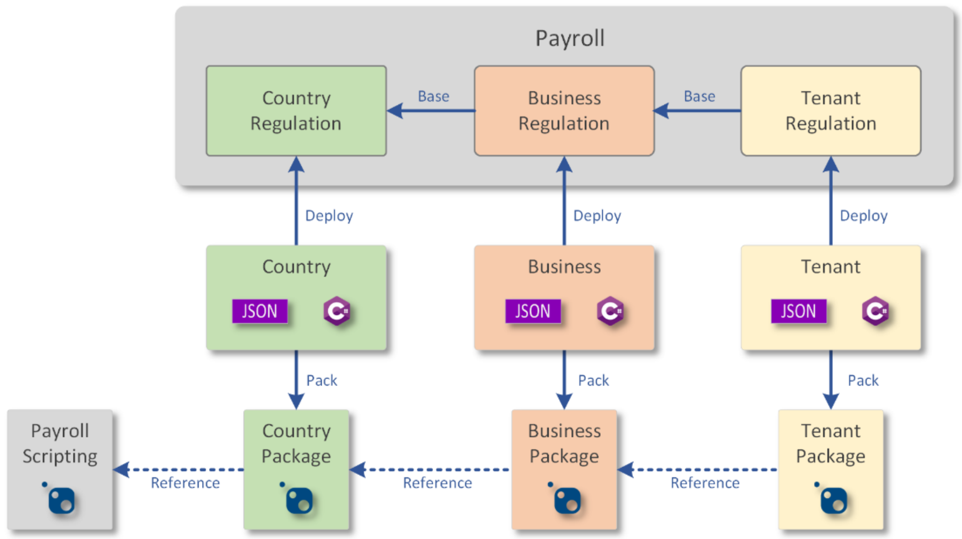


Various tools are available for editing the model (e.g., import and export), which also work for reduced model areas. In addition, there are classes that take over the sequence of model processing and enable the user to access objects directly.

7.1.2 Developing and providing regulations

Regulations are described in JSON files and contain the programmed payroll logic as C# code. This is defined as an embedded object expression or in a source code file (*.cs*). The outsourcing of program logic to source code files is required for shared regulations. In addition to documentation, the NuGet mechanism is used to provide versioned regulations.

The following figure shows the use and provision of regulation packages, as well as the provision of regulations for the payroll system:



8 Clients

Based on the client services, the following clients exist for the backend:

- Payroll console: Importing, exporting, and testing payrolls
- Web application: Visual management of the payroll engine

8.1 Payroll Console

The Payroll Console application offers various client services functions as console commands:

- Payroll import (JSON)
- Import of text tables (CSV, tab-delimited)
- Payroll export (JSON)
- Payroll Test (JSON: Import Payroll, Payrun, Result Test, Cleanup)
- Employee Test (JSON: Import Case Data, Payrun, Result Test)
- Scripting Import, Export, and Rebuild
- Payroll log trail
- Payroll API queries
- Payroll Report in Various Formats

Commands are controlled via command line parameters. For batch execution, command chains can be defined in command files with the extension *.pecmd*. In addition to the predefined commands, it is possible to develop and integrate your own commands.

8.2 Web application

The web application offers various client services functions in the web browser:

- Management of tenants and employees
- Management of regulations
- Recording of case data
- Starting Payruns
- Evaluation of payrun data
- Management of user tasks and developer logs

9 Further information

Further information on the Payroll Engine:

• Website	*https://payrollengine.org/*
• GitHub	*https://github.com/Payroll-Engine*
Wiki	***https://github.com/Payroll-Engine/PayrollEngine/wiki***
Releases	***https://github.com/Payroll-Engine/PayrollEngine/releases***
• NuGet	***https://www.nuget.org/profiles/PayrollEngine***