PAYROLL ENGINE

Payroll Engine Whitepaper

PayrollEngine.org
Beta Release 02/24/2025

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 payroll data of companies. The engine differs from conventional payroll systems in the following points:

Case Driven	Data changes are made on a case-by-case basis and are assigned to the time axis. This means that the business data is valid at any point in time. Automated payruns including retro and forecast scenarios are possible at any time.
Globalization	Per client, the business data can be settled with the respective country- and company-specific laws and local regulations.
Regulations	The application of wages is determined in regulations, which build on each other as components in layers. regulations exist for countries, industries and companies, such as generally binding collective labor agreements or regulations for pension funds.
Customer scaling	Customer customizations are also implemented as regulations, enabling scalable solutions for small to large enterprises.
Automation	The engine has a configurable and extensible data and processing model defined via the REST API. The runtime behavior of the payroll application is controlled via the Scripting API.

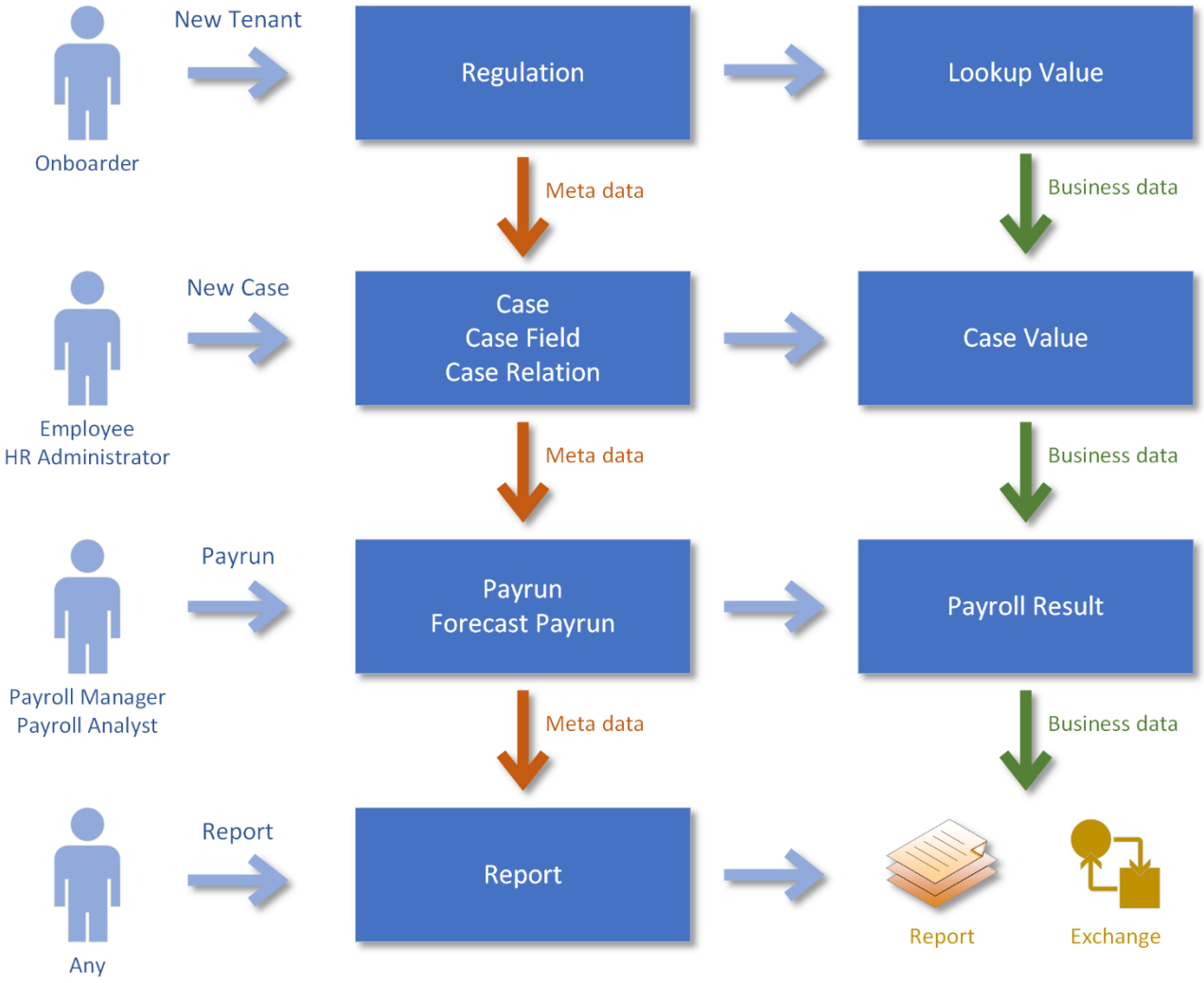
2 Payroll Model

2.1 Core Use Cases

The core payroll use cases cover the following areas:

- Onboarding: setting up the tenant
- Cases Management: Change of company and employee data
- Payrun: Cyclical and non-cyclical calculation of employee wages
- Report: Evaluation of wage data

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

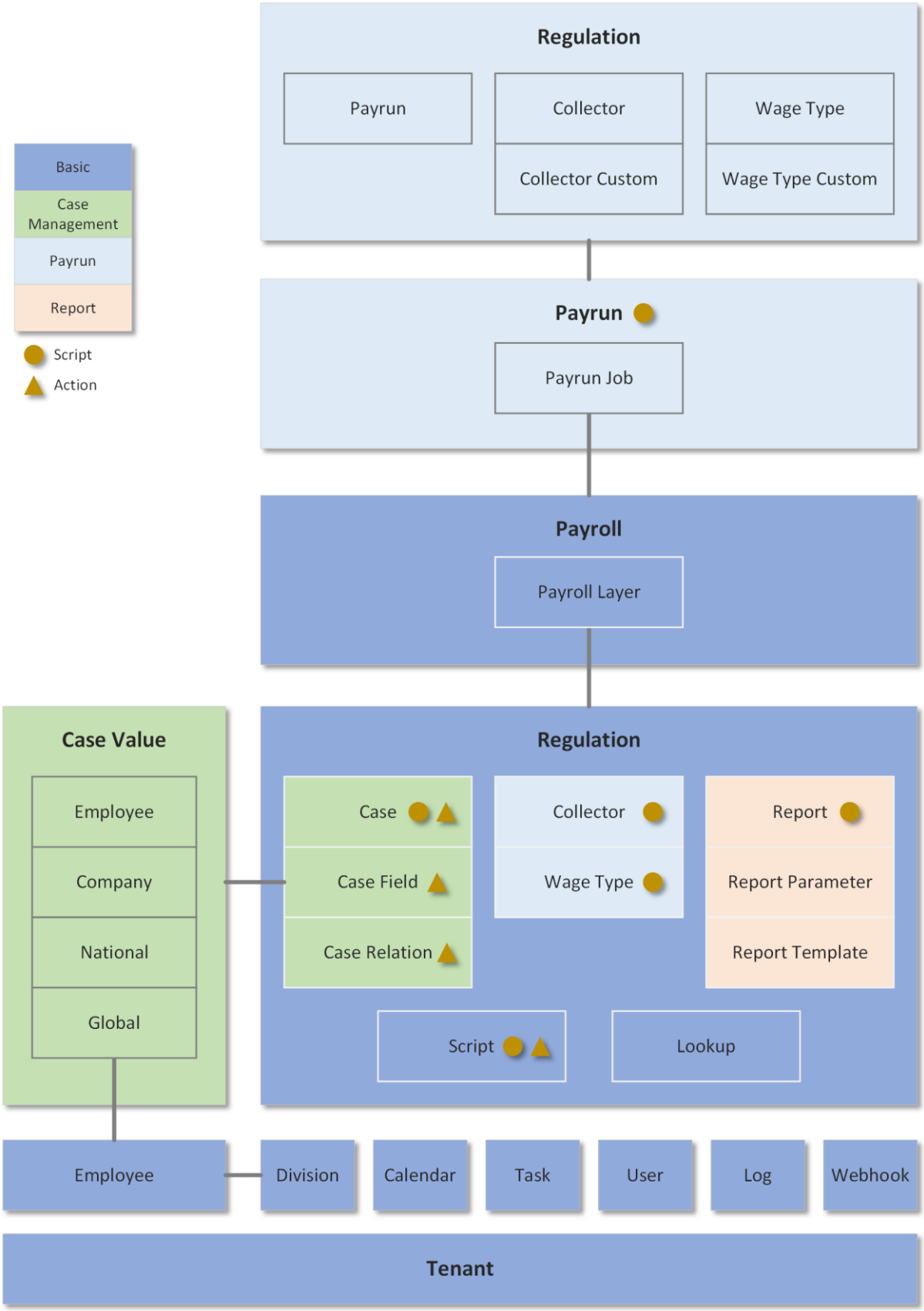


2.2 Model Overview

The payroll model contains objects for the following functional areas:

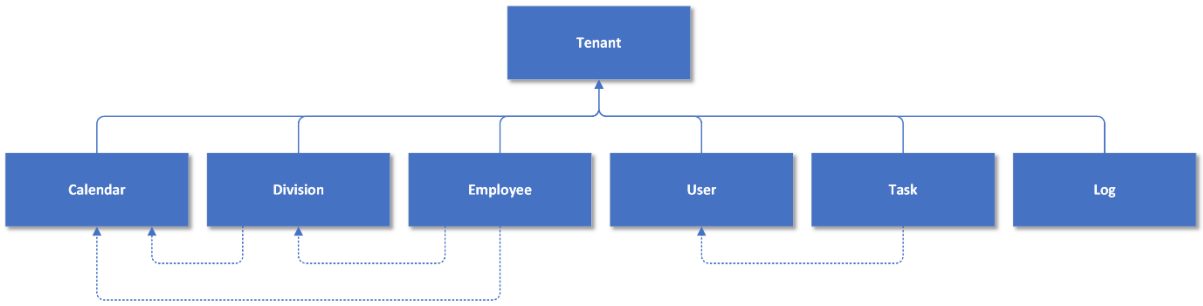
- Base and Infrastructure Objects (*Basic*)
- Objects for recording business data (*case management)*
- Objects for processing wage data (*Payrun*)
- Objects for transformation of output data (*report*)

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



2.3 Client

The client-based payroll service provides the following objects for defining the client with its organizations, employees and users:

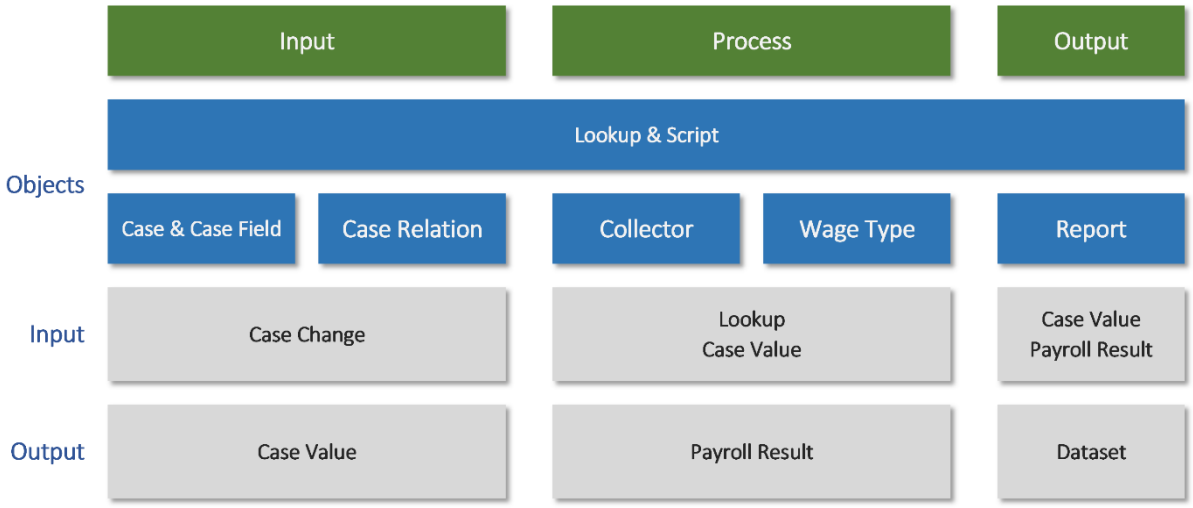


Tenant	Contains all tenant data. All tenant data refer to this object.
Calendar	The client's payroll calendars.
Division	Divides the company into divisions. Each client consists of at least one division.
Employee	Employees assigned to one or more divisions.
User	Users of the system.
Task	User tasks.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

Except for the shared regulations (see Shared Regulations), all domain objects described below are assigned to the client.

2.4 Regulation

A regulation includes the definition of the wage with the elements of wage data input, processing and output:

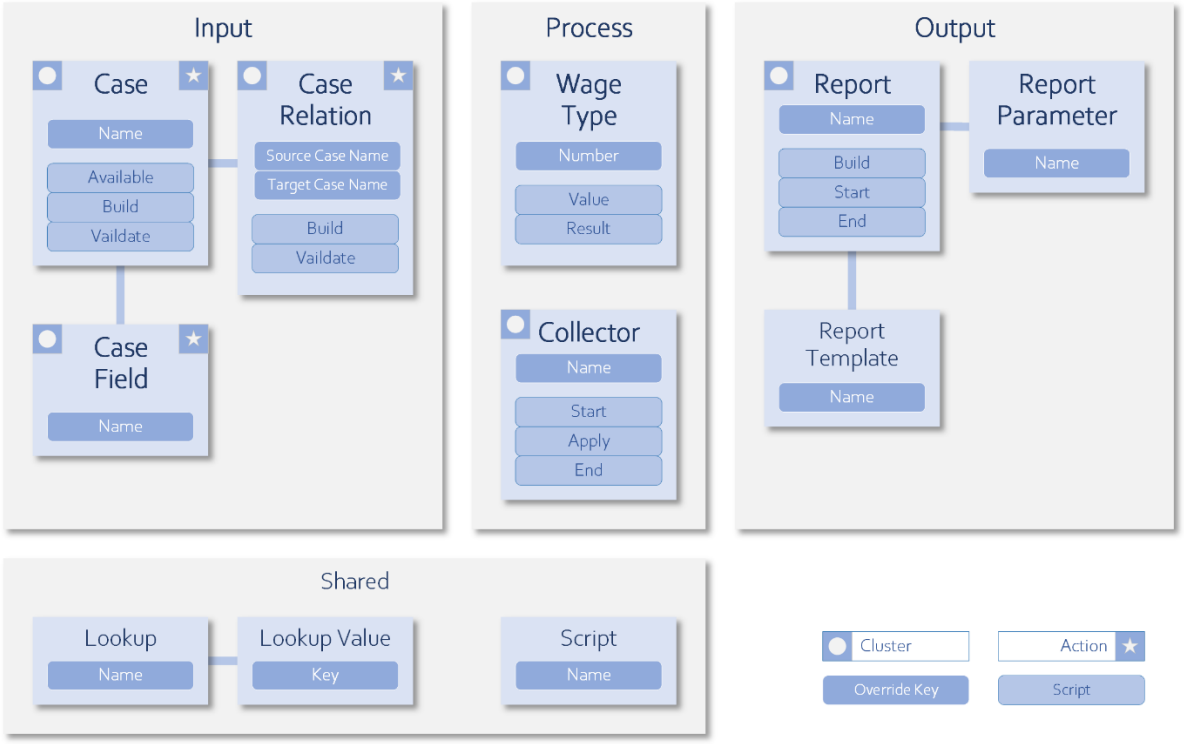


The regulations addresses the following topics:

- Definition of the data model and business rules, see Case Fields
- Calculation of wage data, see Payrun
- Evaluation of company and employee data, see Report
- Testing of company and employee data, see Payroll Tests
- Extend existing regulations, see Regulation
- Sharing software between clients, see Shared Regulations
- Provide software, see Regulations Development and

2.4.1 Regulation Objects

The regulations include 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 global, national, corporate and employee.
Case Field	Field describing the value of the case. The employee data can apply to one or all Divisions.
Case Relation	Controls the dependency between two cases.
Wage Type	Determines the wage type and its calculation. The wage types are calculated in the sequence of the wage type number with a decimal value (e.g., 1000). Subordinate wage types can be included in the processing sequence 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 for 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 run individually or in groups.
Report	Definition of the report.
Report parameters	Parameters of the report.
Report Template	Report template in one language.

Overrides can be used to override a set of regulation object by a higher-level industry or customer regulation (see Payroll). The object to be overridden is identified via the override key, e.g., the case name.

Cluster objects provide an advanced tagging mechanism to include or exclude similar objects according to application criteria (see Payroll Cluster).

An audit trail exists for each rule set object (see Audit Trail) with corresponding REST endpoints. Changes to a regulation (e.g., annual update) are managed in versions, with each version valid from a specific payroll period. This ensures the correct time data of the payruns for retroactive accounting and forecasting.

2.4.2 Application Scenarios

The following areas of application are possible for the payroll regulations:

- Development of customer-specific regulations in onboarding
- Service provider offers commercial regulations
- Industry solutions such as association solutions
- Country-specific and legal settlements
- Company regulations, local or transnational
- Regulations for tests and further developments

2.4.3 Regulation Inheritance

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

- *Case* Name (key)
- *Case Field* Name
- *Case Relation* Case names of 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

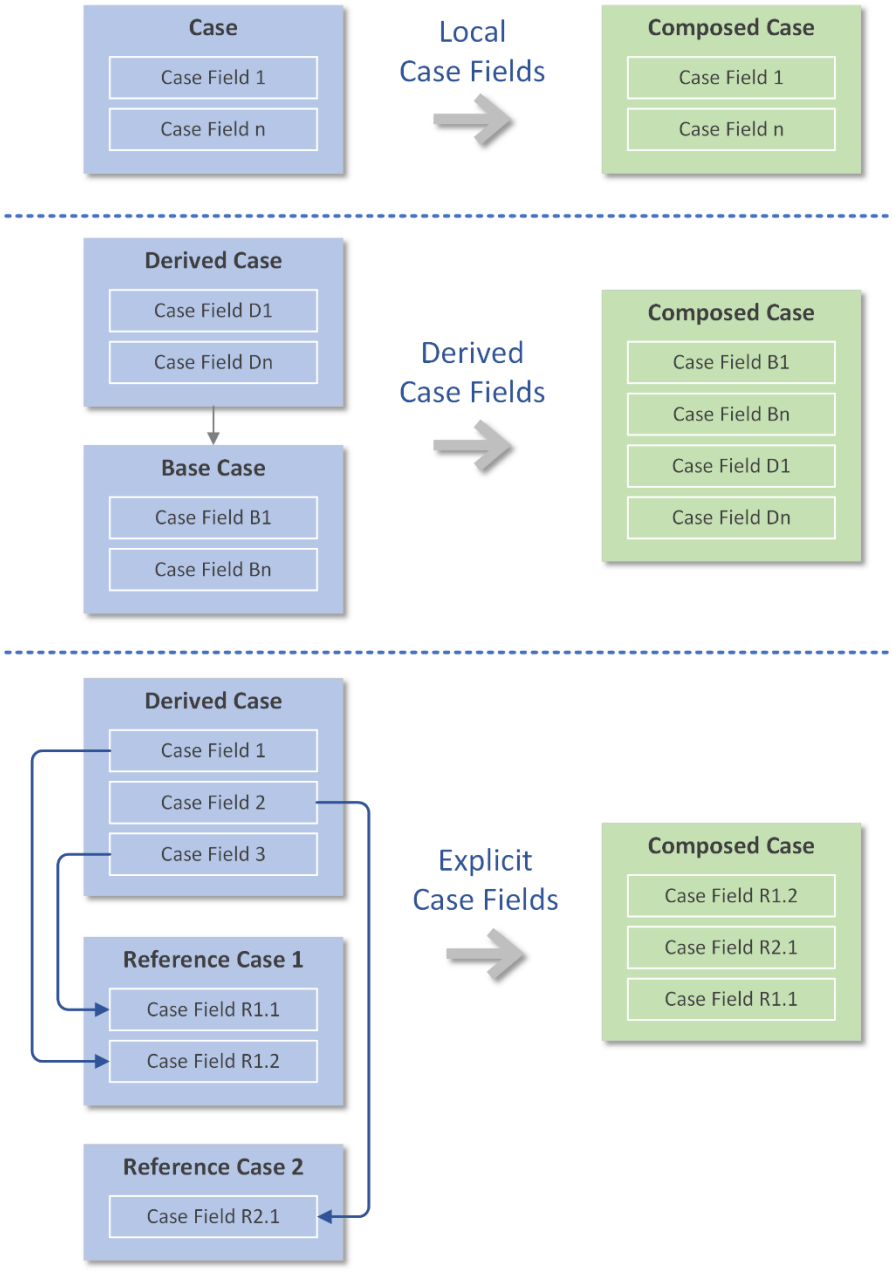
The *Case* object is used to capture case data. The *Case Field* object represents an input field in the case. The *Case Relation object* defines the relationship (availability and value transfer) between two cases.

When entering 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 mutations or the scheduling of wage adjustments (see Time types). This feature distinguishes the Payroll Engine significantly from conventional software solutions, which usually overwrite existing data. The start and end date of the value period can be narrowed down using the Case field date types.

The engine offers different approaches for composing cases and their fields:

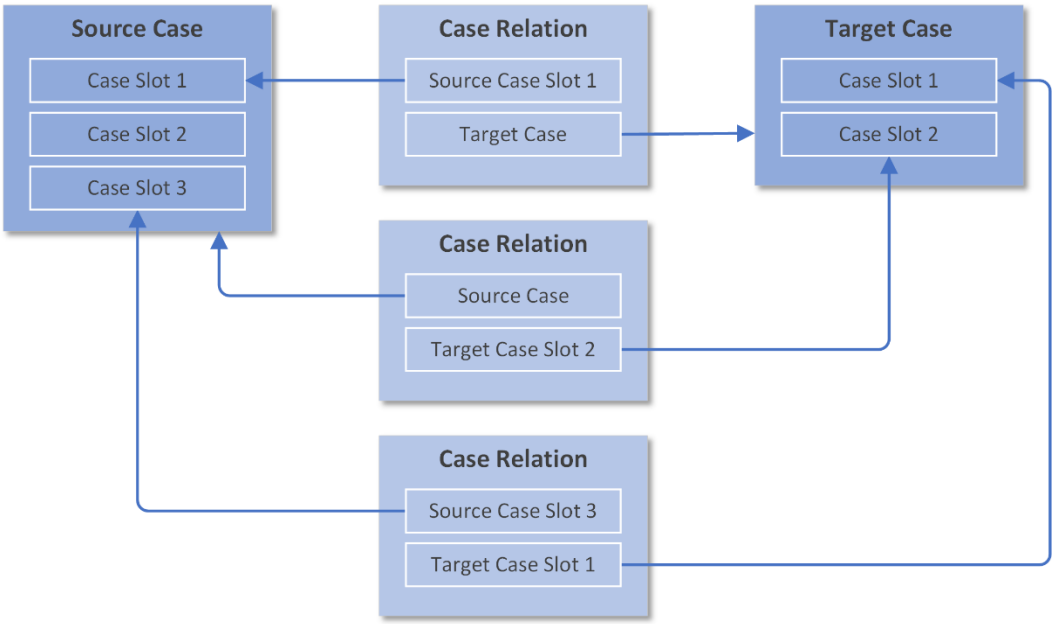
- Declaration of the case field in the case as an integral part
- Transfer of case fields from another case
- Inclusion of case fields from different cases

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



2.4.5 Case Slots

To be able to store multiple 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 of a regulations for

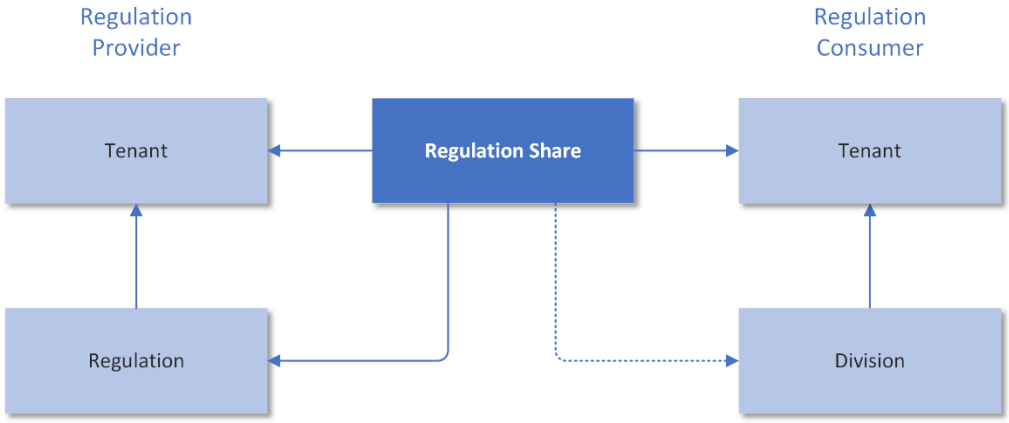
- externally determined data from external places, e.g., a control table.
- Unchangeable selection values that do not change during a payroll period.
- Values that can change cyclically or periodically.

A Lookup consists of a list of values (*Lookup Value*), which is composed 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 via the key, the lookup value can also be searched for via a valence (decimal number). This can be used, e.g., for income dependent tax tables.

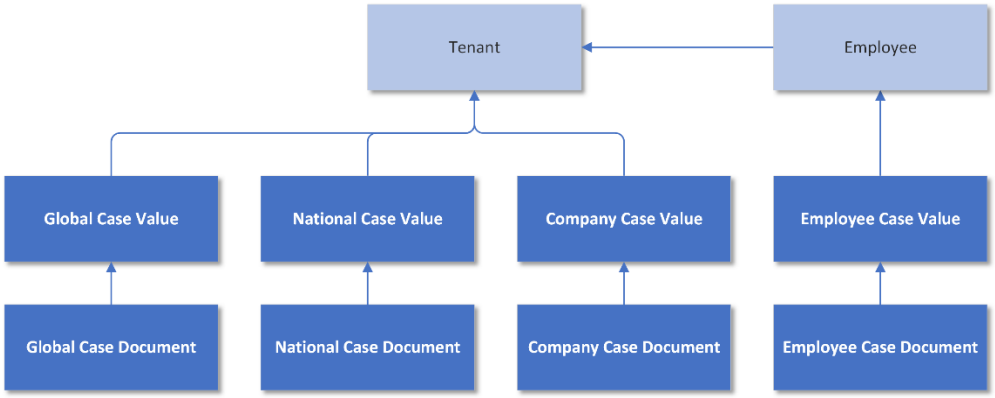
2.5 Shared Regulations

The engine offers the possibility to release a regulation for several clients. This enables the payroll provider to centrally administer country and industry rule sets. Authorizations are used to define which regulations are available for which clients (optionally Divisions):



2.6 Case Data

For each field of a case (*Case Field*) a value (Case *Value)* can be recorded by a 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	National Data.
National Case Document	The National Documents.
Company Case Value	Company data.
Company Case Document	Corporate documents.
Employee Case Value	Employee data.
Employee Case Document	Employee documents.

A case value is available either for all Divisions or for one Division. The definition is made in the data field (*Case Field*) of the regulations.

The case data is unchangeable and is supplemented by the case changes. The data of a wage period can be determined based on this change history.

The value of a case (*Case Value*) can be limited to a validity period (start/end) (see Time types). The payrun only takes the value into account if its period matches the wage period.

Case types can be used in the regulations as follows:

Regulation	Global	National	Company	Employees
Client	✓	✓	✓	✓
Business	✓	✓	✓	✓
National	✓	✓		
Global	✓			

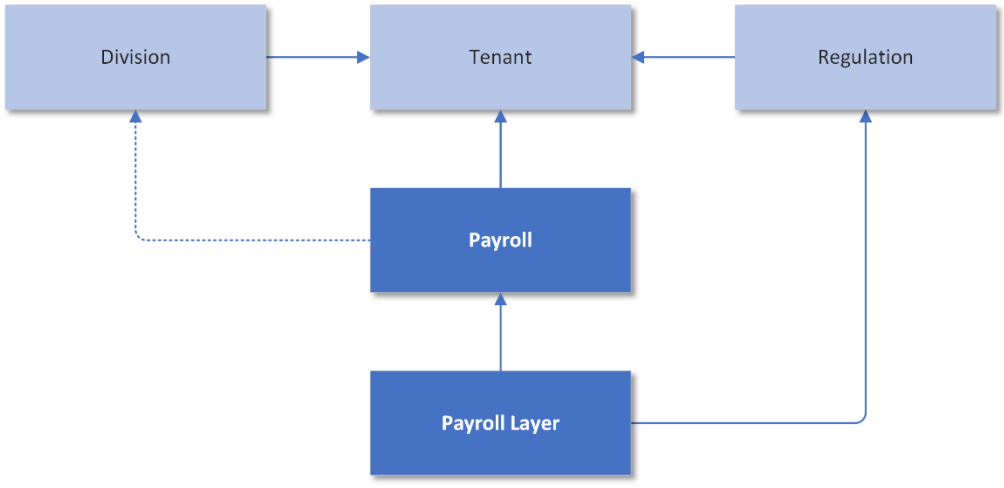
2.6.1 Case Cancel

A case that allows cancellations controls the reset behavior for numeric case values via the cancellation type:

- Take over previous value
- Initialize value to default
- Invert value
- Maintain value

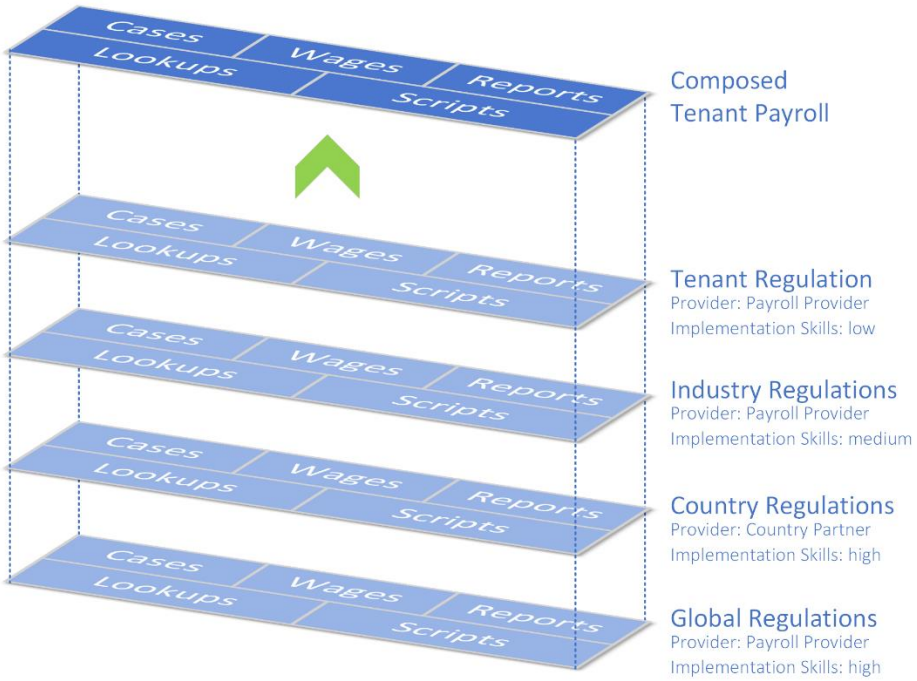
2.7 Payroll

The payroll object links multiple regulations via a layered model to create a virtual payroll model. The following objects are involved:



Payroll	Combines several layers of regulations into one payroll and is assigned to one business unit.
Payroll Layer	Determines the priorities and order of the regulations.

Payroll provides a consolidated view of the regulations:



2.7.1 Payroll Layer

The evaluation sequence of the regulations is defined in the *Payroll Layer* via the level (1st sort criterion) and the priority (2nd sort criterion). The number of layers is unlimited. The following example shows three rule set layers linked to 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

The overlay key can be used to override objects from lower layers. In this case, the values of all layers are dynamically combined into one object:

- Localizations
- Attributes

- Clusters
- Case Slots, Lookups and Actions
- Text fields

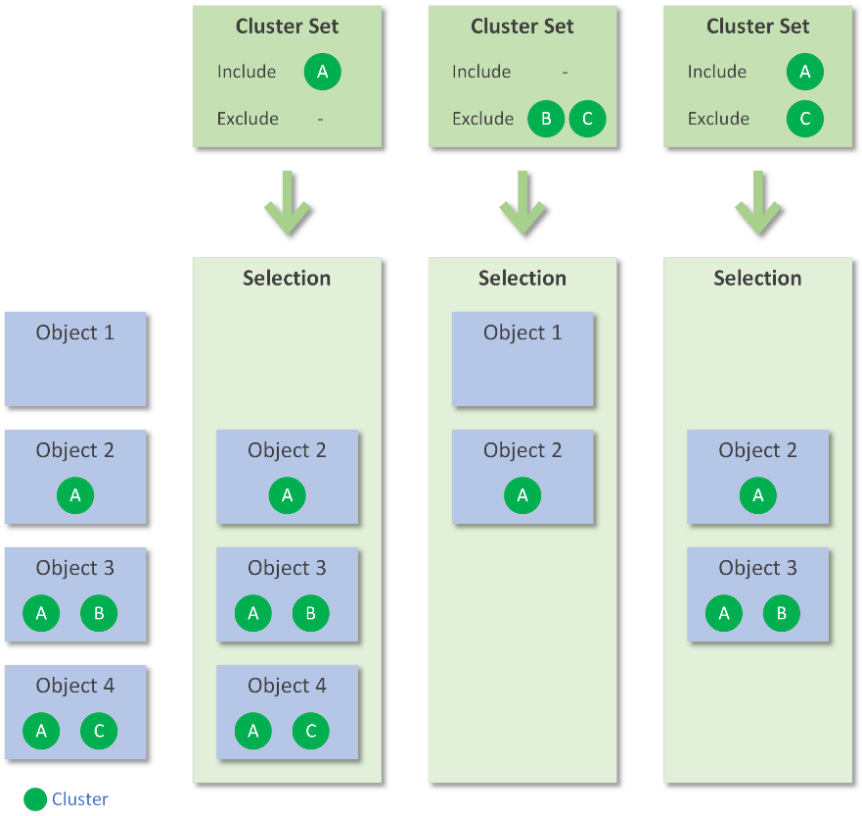
2.7.3 Payroll Clusters

Using clusters, rule set objects can be grouped according to any criteria. This can be used to:

- filter API queries e.g., for grouping cases.
- optimize the payrun by reducing case data.
- limit the payroll results.

Several cluster names (analogous to tags) can be assigned to each cluster object. The 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 included can be further restricted.

The following figure shows different scenarios for the selection of cluster objects:



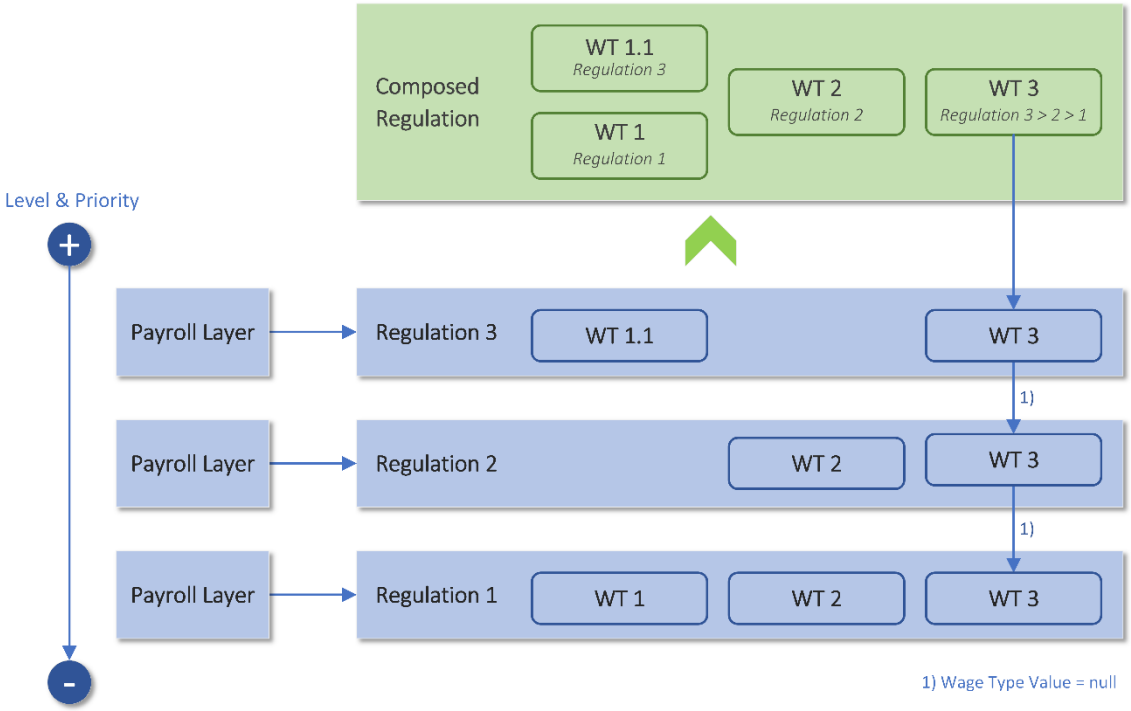
Cluster sets are defined in the *payroll* object and can be used in the payrun. The following cluster sets exist:

- *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 retro payrun
- *Wage Type Cluster Set*: available wage types in the payrun
- *Wage Type Retro Cluster Set*: available wage types in retro payrun
- *Wage Type Period Cluster Set*: Generate Period Results from Case Data
- *Case Value Cluster Set*: available case data in the payrun for additional results

2.7.4 Payroll Scripting

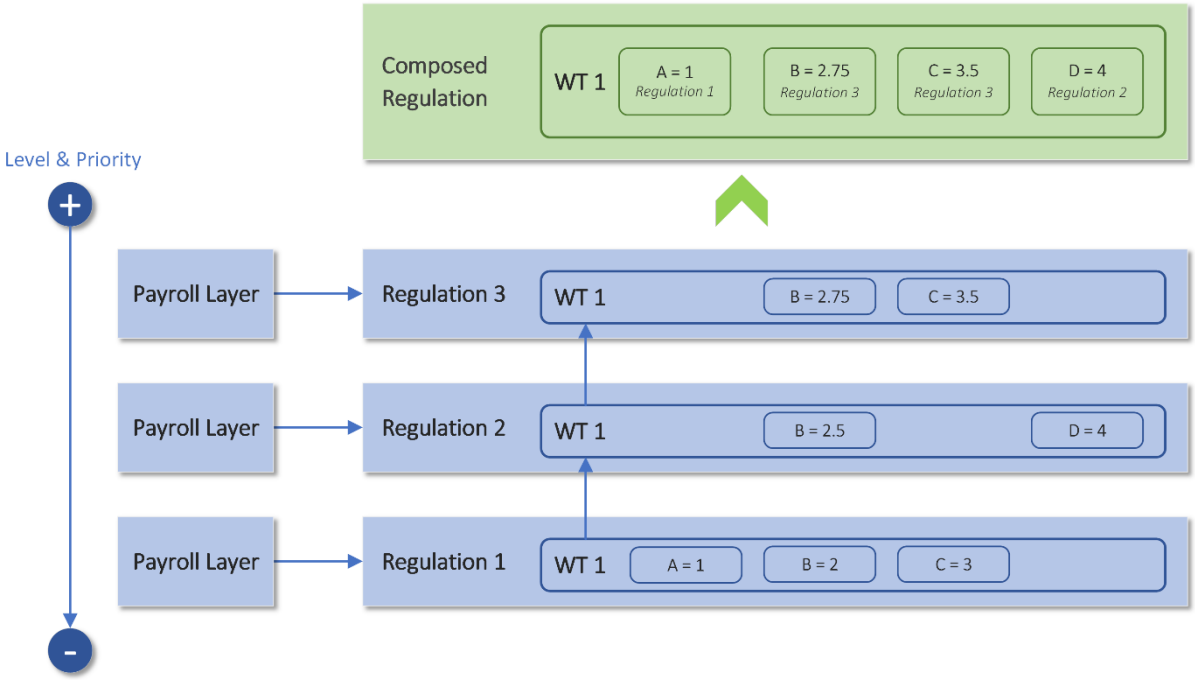
The Scripting API allows controlling the runtime behavior for specific payroll objects (see Scripting API). The scripts are evaluated using the evaluation sequence in the payroll layer (see Payroll Scripting).

The following example shows the calculation of wage types:



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

In contrast, object attributes are evaluated in the opposite direction, so that the value can be overridden by a higher-level regulation:



2.8 Wage Calculation

2.8.1 Wage types

Employees' 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, which calculates the consecutive wage bases. The number of wage types is not limited and results from the sum of the payroll regulations.

2.8.2 Collectors

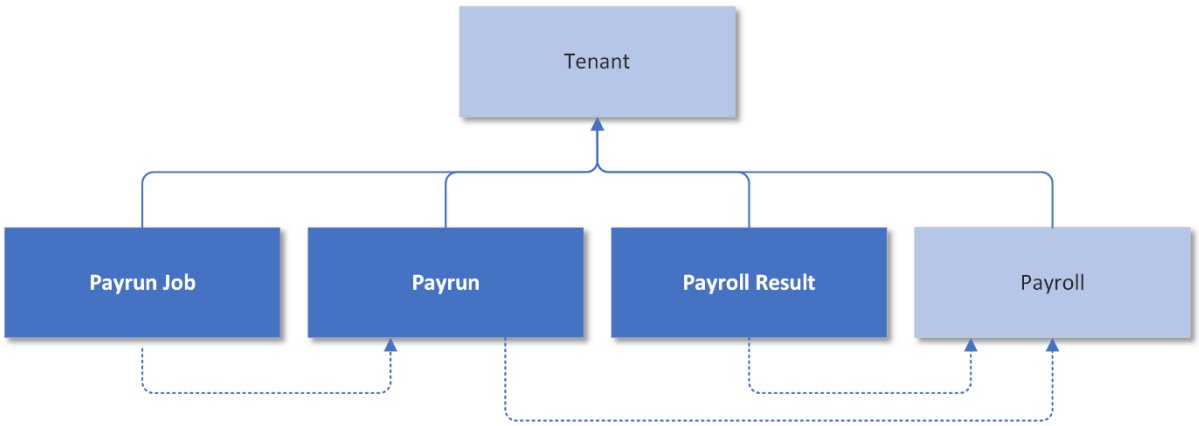
Collectors collect the values of wage types together and provide various predefined operators:

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

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

2.9 Payrun

Wage data is calculated using the following objects:



Payrun Job	The 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 Regulations Scripting chapter.

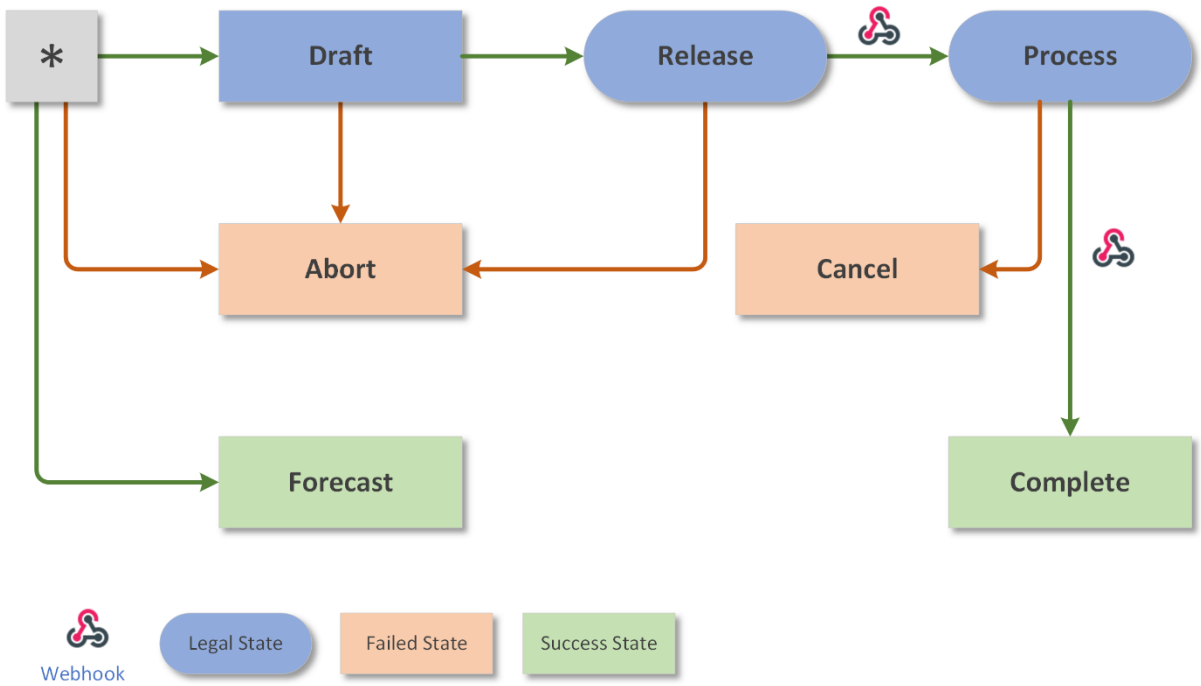
2.9.1 Payrun Job

The *Payrun* Job starts the payrun for a pay period and stores the results in the *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:

- Legal wage declaration
- Forecast analysis of wage data for forecasts, case scenarios, etc.

The payrun job is controlled by the job status:



Job Status	Type	Description	Webhook
*		New payrun job	
Draft	Working	Legal payrun job to 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	Forecast Payrun Job	
Abort	Final	Legal payrun job canceled before release	
Cancel	Final	Legal payrun job processed incorrectly	*PayrunJobFinish*

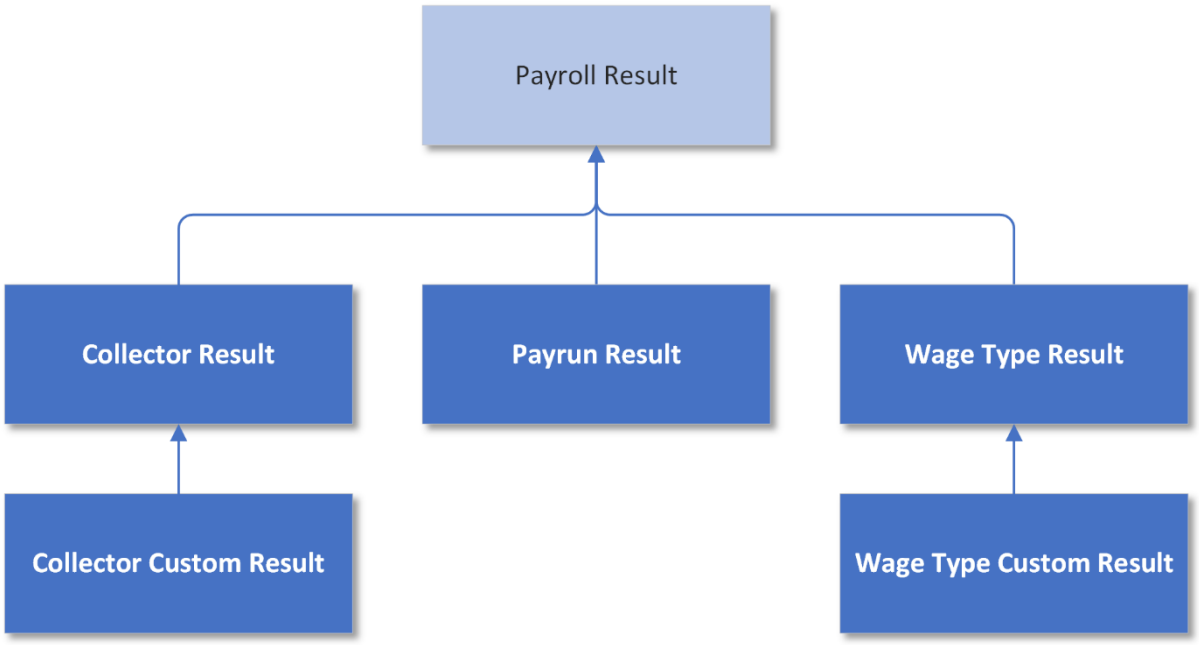
For the statutory payruns, there can only be one job with *Draft* status per payrun type (*Payrun*) and wage period. Multiple jobs with the same status are possible in the *Release* and *Process* statuses. Forecast payruns can be executed as often as required.

2.9.2 Payrun Restart

During the payrun, all wage types are processed in the order of the wage type numbers. In special cases, the payrun can be restarted for an employee. Each run is identified in the wage type by the run counter. With the help of the run time values (see Payrun Scripting), data can be exchanged between the runs.

2.9.3 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	Payrun-specific result (also case 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 a payroll, cluster sets are used (see Payroll Cluster) to control which results are to be generated.

2.10 Special Payruns

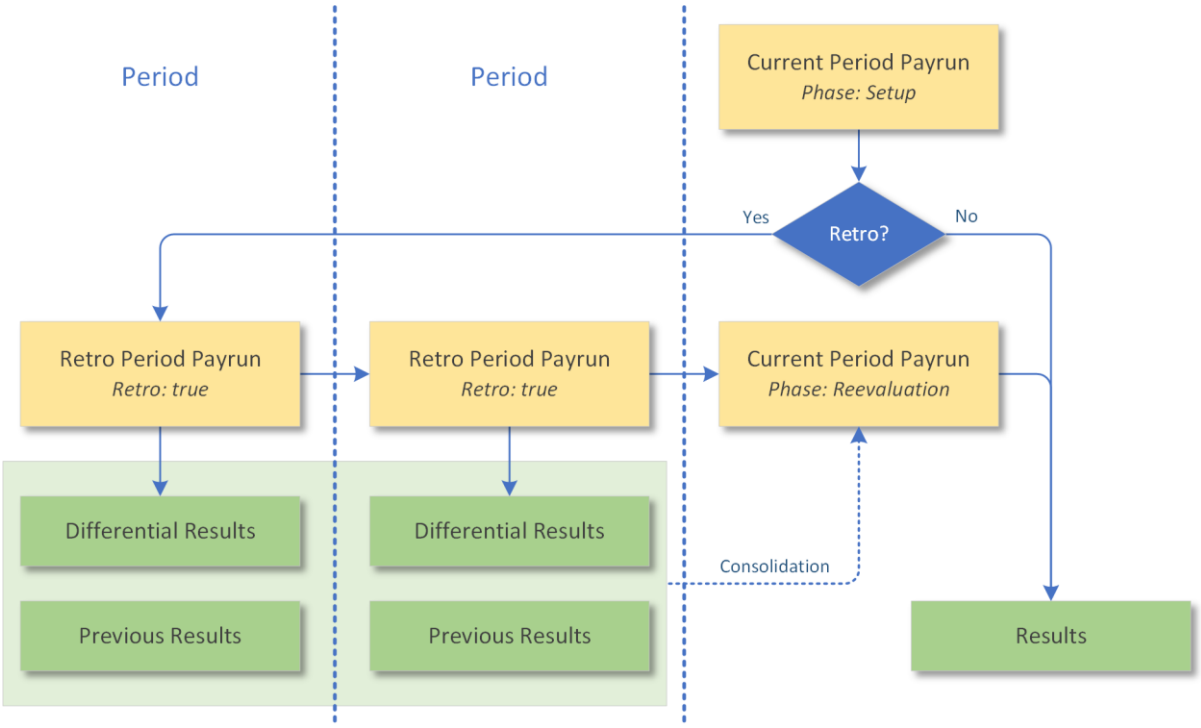
2.10.1 Incremental Payrun

If multiple payruns are performed within a period, the engine stores only incremental results, i.e., values that have changed since the last payrun. To determine the currently valid (consolidated) results of a wage period, the REST API provides corresponding endpoints.

2.10.2 Retro Calculation

A retro payrun is performed automatically if new mutations have occurred since the last payrun that affect past pay periods.

The retro payrun process:



The steps of the back calculation are:

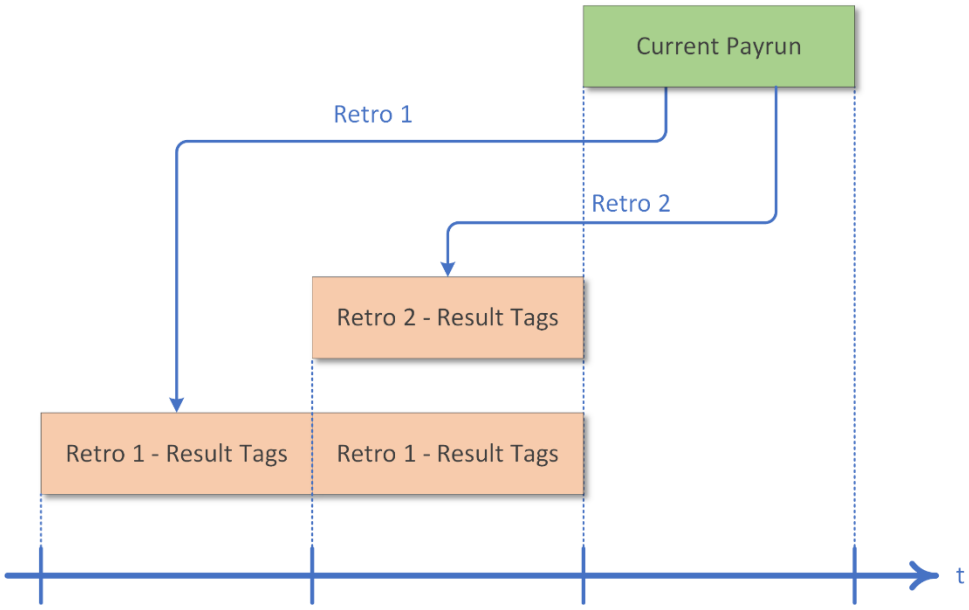
1. Calculation of the current pay period (with mutation to the previous period).
   → Execution phase *setup*
   → Transient results
2. Calculation of all previous periods starting from the mutation period to the previous period.
   → Save incremental results of the previous periods
3. Recalculation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

For forecasts, the foreruns with the same forecast name are valid. The number of periods in the recalculation is unlimited, but it is possible to limit the recalculation to the payroll cycle.

2.10.3 Manual recalculation

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

In the following scenario, the current period's payrun triggers two retroactive accounting payruns:



2.11 Forecast

Two interventions are required to create forecasts:

- Change of case data for a forecast
- Running a payrun job for a forecast

If the payrun is started for a forecast, all case data of the forecast are available as well as all other case data that are not assigned to a forecast. The normal payrun, however, ignores the case data of forecasts.

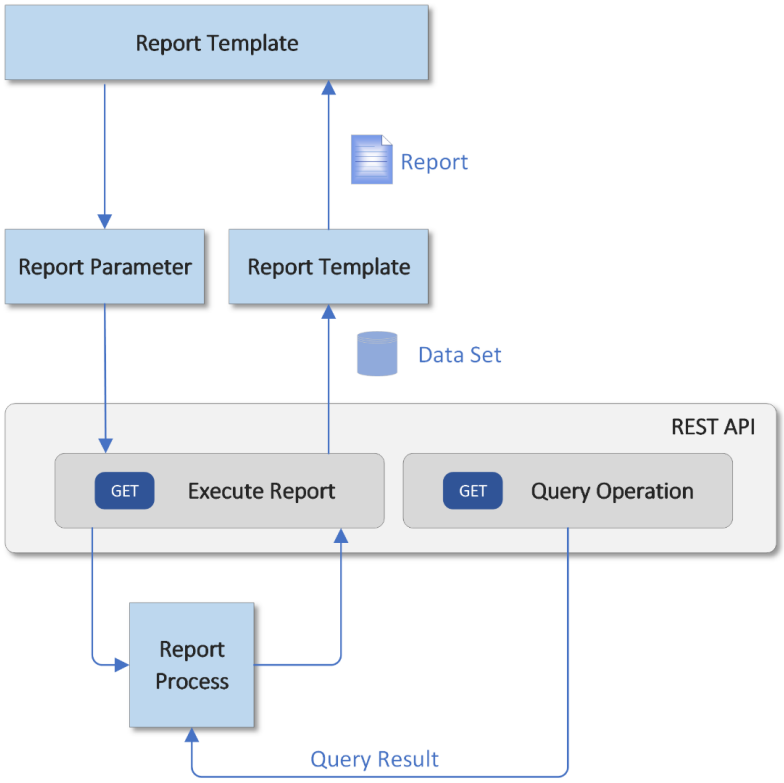
2.12 Report

Report generation is done in three steps, each step being optional:

1. Report Preparation: *Report Build*
2. Report queries: *Report Start*
3. Editing or post-processing of the 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. To determine the report data, all GET endpoints of the REST API serve as a data source.

Report Generation:

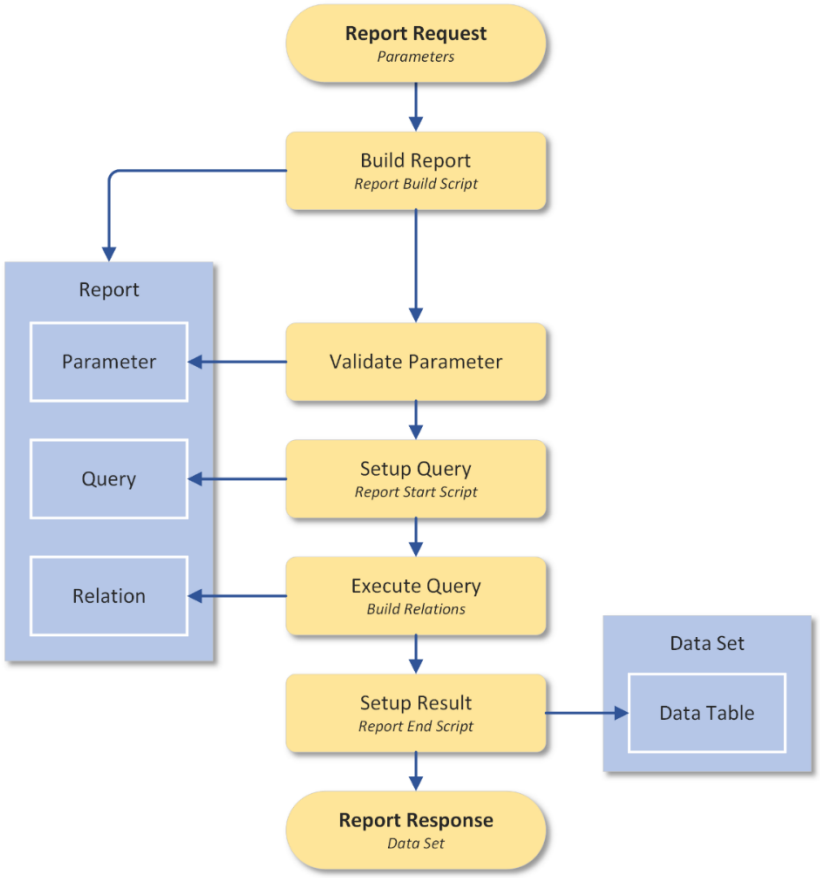


The language dependent report layout (*ReportTemplate*) is irrelevant for the calculation of the report data and is determined by the REST client (*.frx, .docx, .rdlc, ...)*.

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 Pocess

The creation of a report is done in the following steps:



1. Report creation: Report parameters can be customized in the *Report BuildScript.*
2. Verification and completion of the report parameters.
3. Preparation of the *queries*: The queries are predefined in the report and can be customized in the *report start script.*
4. Query of the report data: The report can execute multiple queries, storing the data of an API endpoint (GET) 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 adapted 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 only of queries. A complex report can contain all the post-processing logic.

Other Report Functions:

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

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 the reporting status.

3 Basic concepts

3.1 Languages

The payroll service supports all RFC 4646 languages with English as the translation base. For references between objects the English identifier is used.

3.2 Audit Trail

For the central payroll data, the engine logs every change in the audit trail. This applies to all regulation objects and case data. The audit trail of the case data also serves as the basis for calculating the 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	
Decimal, Money, Percent, Hour, Day, Week, Month, Year, Distance, NumericBoolean	Number	Decimal	
Boolean	Boolean	Boolean	
None	Zero	zero	

3.4 Calendar

The calendar divides the dates into two time-units:

- **period** in which the salary is paid, usually the calendar month
- **Cycle** in which the salary periods repeat, 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:

Period ↓ / Cycle →	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 cyclic 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 a month with an average value
- Definition of the days in the working week (e.g., Monday to Friday)

The calendar is determined by the following objects and can be overridden at different levels. In the evaluation priority there are the following calendar users:

1. Wage type (temporary in the payrun)
2. Employees
3. Division
4. Client

3.4.2 World Time

REST API date values must be specified in UTC, other time models will cause errors.

3.4.3 Time types

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

Type	Description	Periods start	Periods end	Number Period values
Timeless	Value without time commitment e.g., precautionary duty	-	-	1
Moment	Value that is assigned to a point in time e.g., bonus	x	-	1..n Total

Period	Value for a period e.g., working hours	x	(x)	n overlaying
CalendarPeriod	Value distributed by days on calendar period e.g., monthly wage	x	(x)	n overlaying

3.4.4 Case field date types

The start and end date of a case field can be linked to the following values:

Case filed date type	Description	Notes
Day	The date is a day	Default, no validation check
Sunday ... Saturday	The date is a weekday	
Month	The date is a month	
January ... December	The date is a year-month	
Year	The date is a year	
PeriodStart	The date must match the period start	Exact match
PeriodStartDate	The date must match the period start day	
PeriodEnd	The date must match the period end	Exact match
PeriodEndDate	The date must match the period end day	
CycleStart	The date must match the cycle start	Exact match
CycleStartDate	The date must match the cycle start day	
CycleStart	The date must match the cycle end	Exact match
CycleStartDate	The date must match the cycle end day	

If the start or end date does not match the assigned date type, a validation error will occur when creating the case.

3.5 Object properties

Each API object has the following basic properties:

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

3.6 Object attributes

Object attributes are user-defined data whose content does not affect the engine behavior. 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 the input of case data
- Control/parameterization of scripting functions (collector or wage type)
- Storage of additional wage type results with e.g., reporting data

The attributes are managed in a value list (dictionary) in the object. Some objects provide API endpoints to edit the object attributes directly.

In REST queries, attribute fields can be used for filtering (see Attribute 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

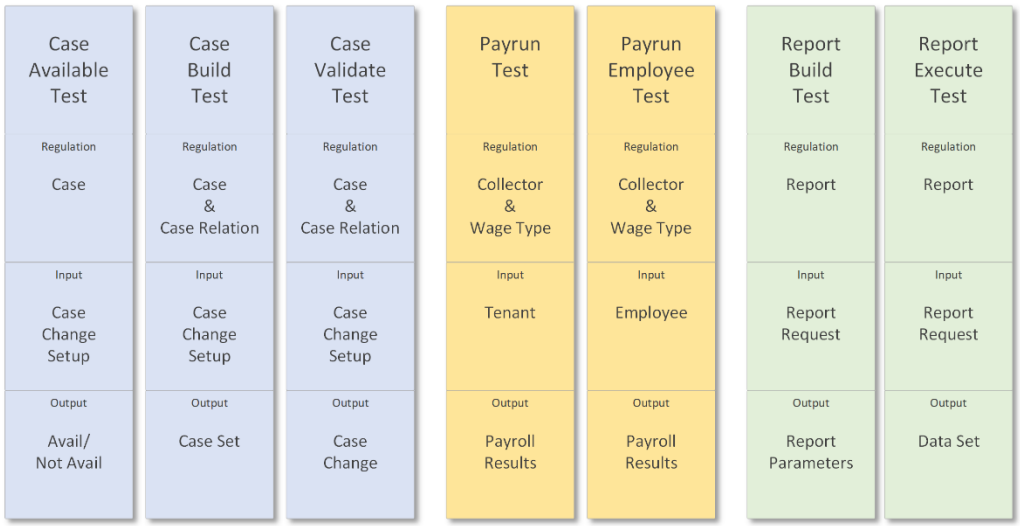
Based on modern approaches in software development, the Payroll Engine also offers test-driven development of payroll calculation. With the help of Client Services (see Client Services), the following aspects of payroll can be tested automatically:

- Employee cases in the past and the future
- Any payruns within a wage period and over several wage periods
- Comparison of the calculated wage data with the expected results

Automated testing is possible in three areas:

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

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



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 Client Payrun Tests

In the client payrun test, a new client is created and the results of the payruns are compared with the expected results. After the test, the client is deleted.

4.2.2 Employee Payrun Tests

For more extensive client scenarios, tests for individual employees are possible. The test of the employee payrun requires an existing client. The payrun is performed on the copy of an existing employee. The cleanup of the test data is done manually.

4.3 Report Tests

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

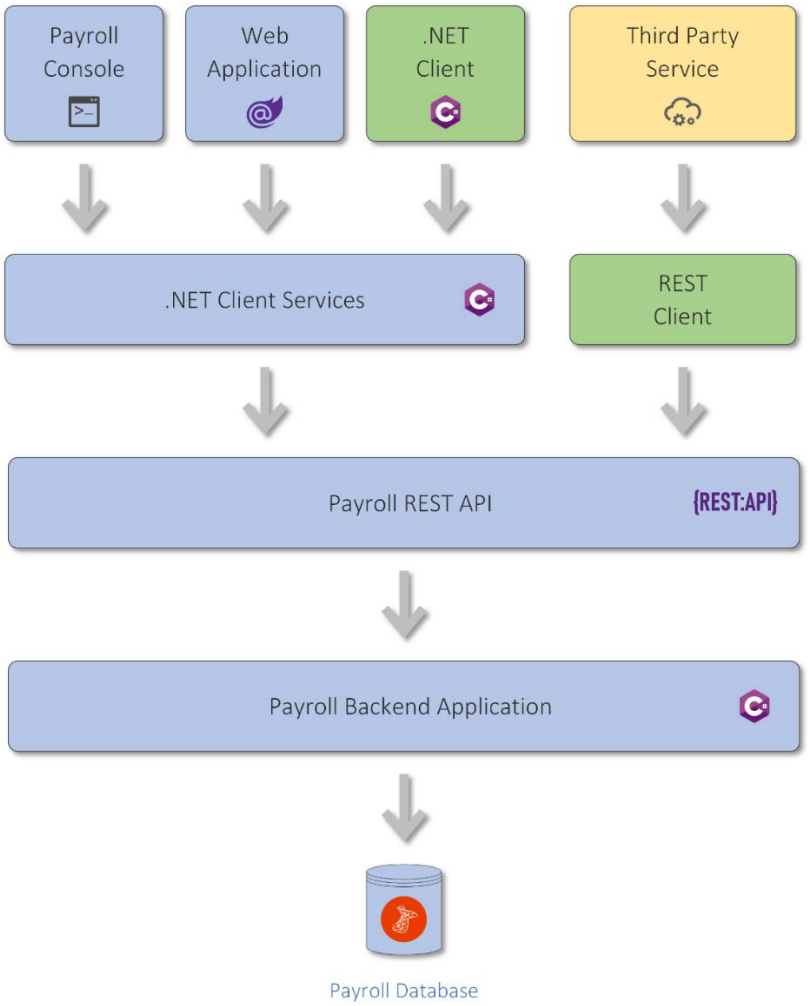
4.4 Test Usage

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 (see Client Services) and can be extended individually.

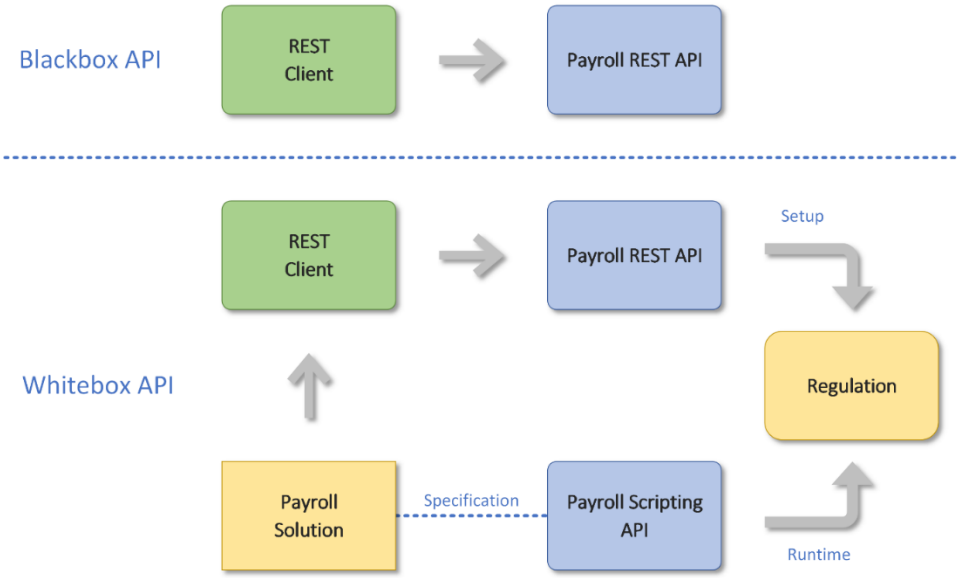
5 Payroll Backend

The Payroll Backend consists of the following components:



5.1 Whitebox API

The Payroll Engine is a Whitebox REST API with an additional interface, the Scripting API, through which the runtime behavior can be influenced. The following figure shows the differences between the Blackbox and the Whitebox REST API:



5.2 Technologies

The following technologies are used in the Payroll Backend:

- REST API — OpenAPI 3.0 (OAS3)
  Media type — application/json
- Scripting API — C# 11
- Server Application — .NET 7 Runtime for Server Apps
- Database — SQL Server 2017

5.3 List Queries

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

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

5.3.1 Attribute Queries

The engine offers the possibility 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*
- Numeric attribute: *NA_*
  → example: *NA_SecurityLimit*

5.4 Webhooks

The engine provides 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 was terminated	*Request: PayrunJob*	✓
ReportFunctionRequest	*Report* function request	*Operation: <Custom>* *Request: <Custom>*	-
TaskChange	Task change	*Request: TaskChange*	

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

5.5 Administration API

The payroll API provides several endpoints for system administration:

- Exit the application
- Cleaning the application cache
- Determine the available report endpoints (see Report)

5.6 Application Logs

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

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

- Log for subcontract runs (see Payrun Logs)
- Log for reports (see Report Logs)

6 Scripting API

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

Case management

- Determine availability of a case
- Structure of a case and determine case values
- Set behavior between cases
- Case validation

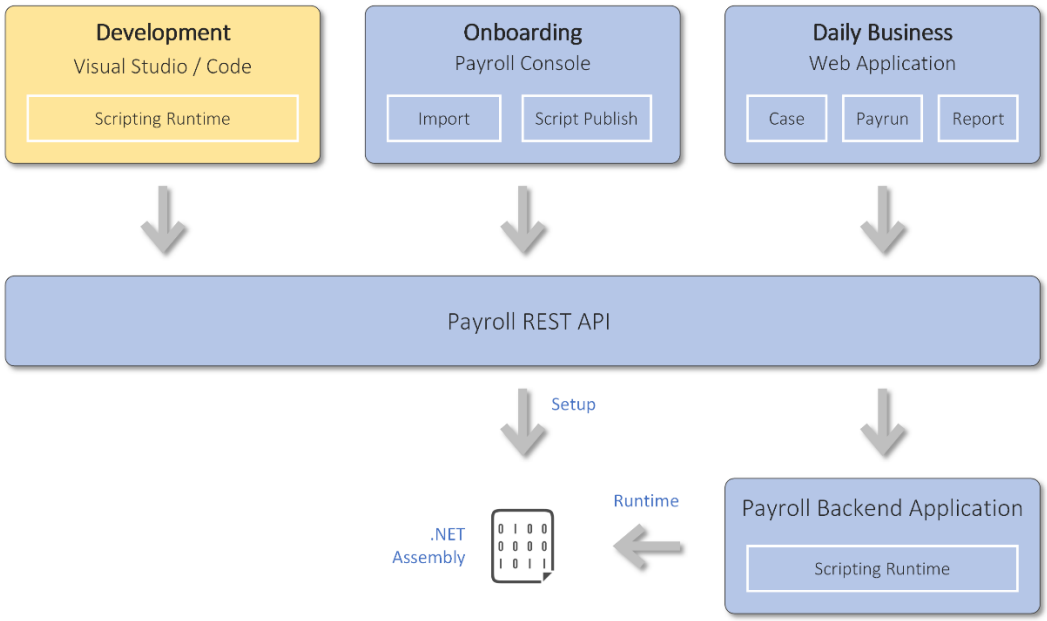
Payrun

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

Report

- Set report parameters
- Calculate report data

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

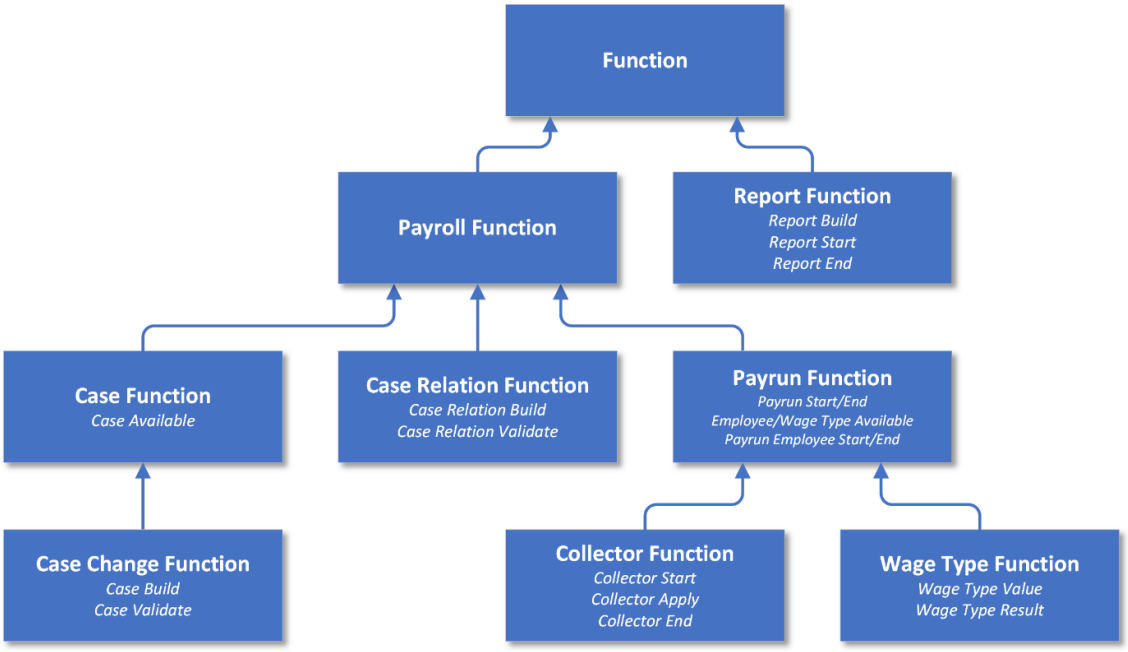


During onboarding, scripts are developed and transferred to the *Payroll* REST API using the Payroll *Console.* When the scripts are onboarded, they are translated (compiled) into machine language. At runtime, the binary scripts are loaded and executed by the payroll process. In the process, the script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

In scripting development, scripts can be tested locally for debugging.

6.1 Functions

A function is a programmed sequence that is called by an Engine process at specific points. Functions are divided into the following sections:



The available scripting functions are:

Object	Function	Description	Type	Overlay
Case	Available	Test if 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 if 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	Apply wage type 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 creation	-	-
Report	End	End of report creation	-	-

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

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

In the chapter Payroll Script both scenarios are shown.

6.1.1 Extended Functions

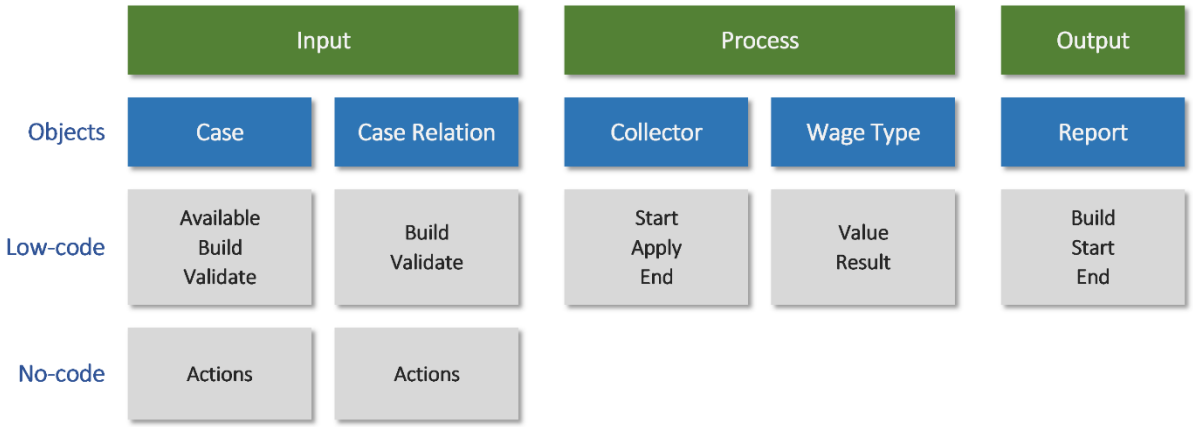
The integrated functions can be extended by scripts. There are two ways to do this in C#:

- Extension Method (Microsoft Extension Methods)
- Partial Classes (Microsoft Partial Classes)

The partial classes are suitable for outsourcing the business code into a separate class model. In case scripting, extension methods offer the possibility to define the generation and validation per case field.

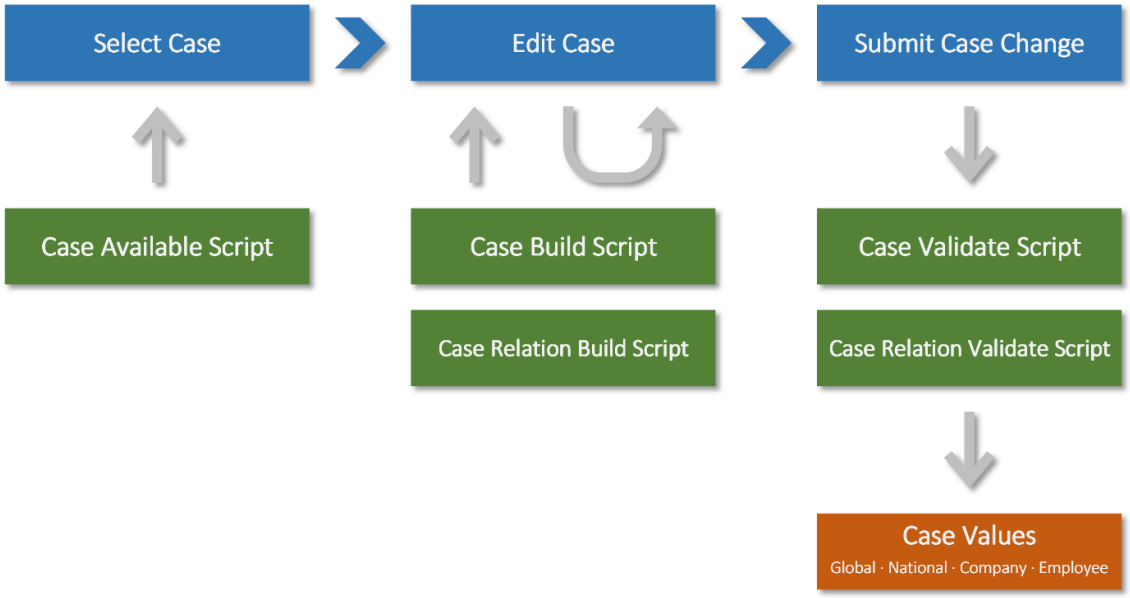
6.2 Regulations Scripting

To control the runtime behavior, payroll provides scripting functions for the rule set objects, which are described using C# Low-Code. For case management, the functions can be controlled via no-code (*actions)*:



6.3 Case Scripting

The life cycle of a wage 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 data is changed. This results in the following processing sequence:

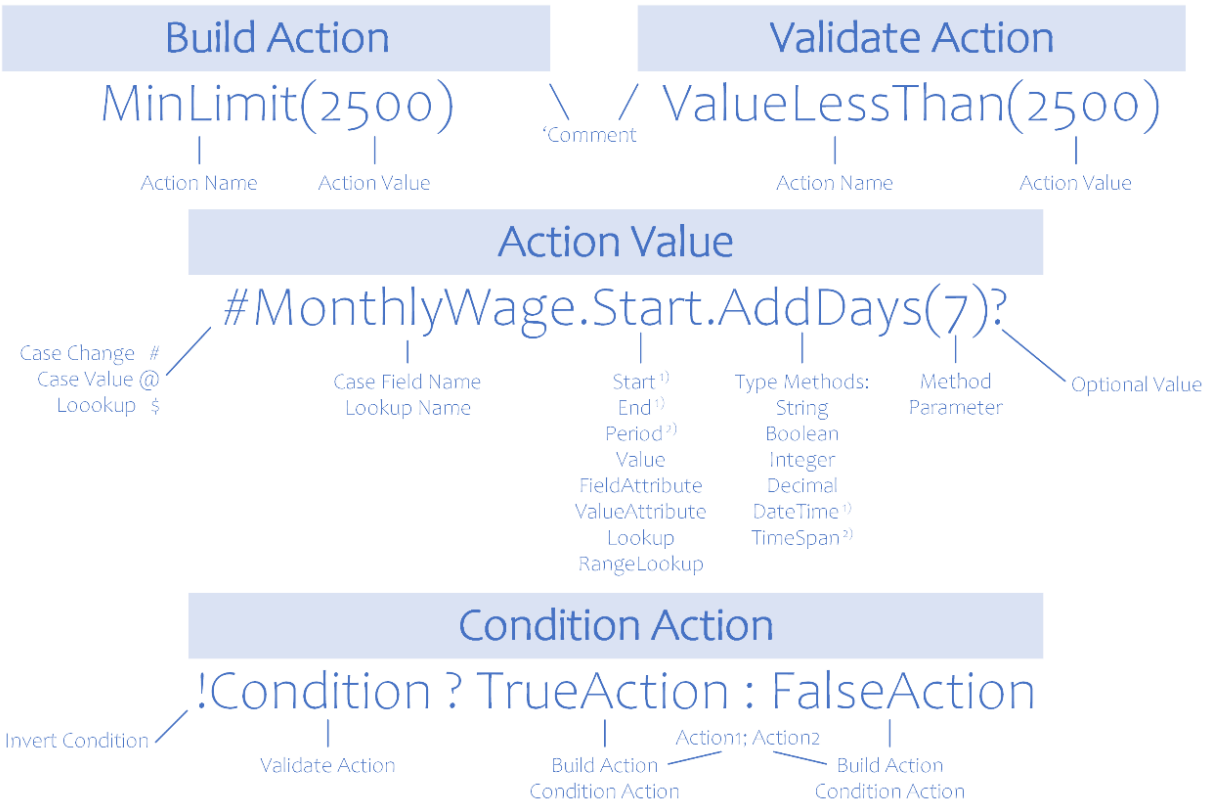


6.3.1 Case Actions

Case Actions provide the ability to control case functions without programming knowledge. Case Actions provide the following functions:

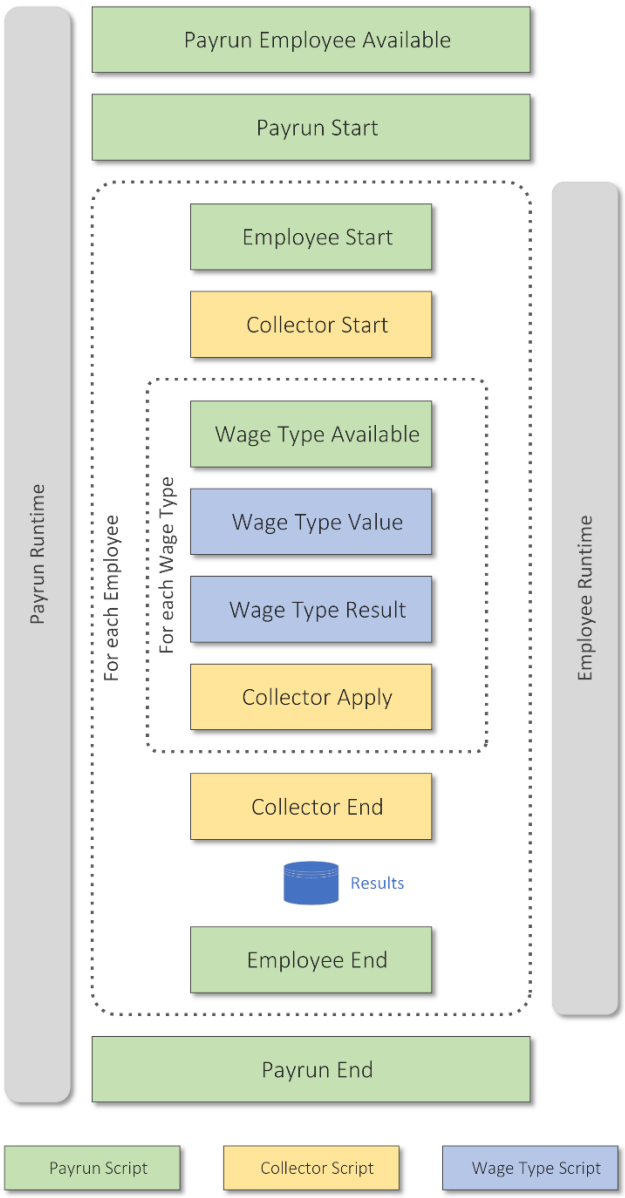
- Access to employee data and lookups
- Set values conditionally
- Check values
- Input field control

The following overview shows the command set of Case Actions:



6.4 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, 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 maturity values:

Function	Payrun maturity values	Employee fair 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*).

6.5 Payrun Time-Value Calculation

When calculating a period value, all relevant mutations within the period are distributed proportionally according to the calendar (e.g., sub-monthly wage adjustments). The Scripting API offers the possibility to use the distribution logic of the calendar, which considerably simplifies the calculation syntax. Two case values with different mutations can be calculated using the usual mathematical operators (plus, minus, multiplication and subtraction).

6.6 Payrun Logs

Log entries can be generated at different levels via scripting functions. The logs are assigned to the client and can be evaluated via the API. A start parameter of the payrun job can be used to specify which log levels are to be logged.

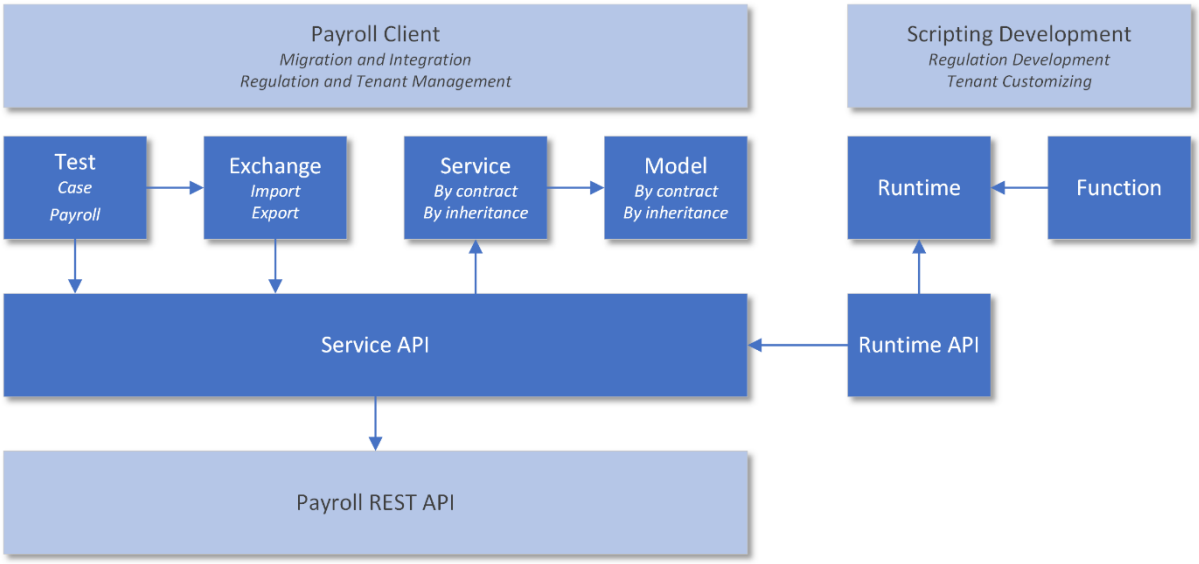
6.7 REST API Scripts

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

7 Client Services

The Client Services are additional services to use the payroll API more efficiently for

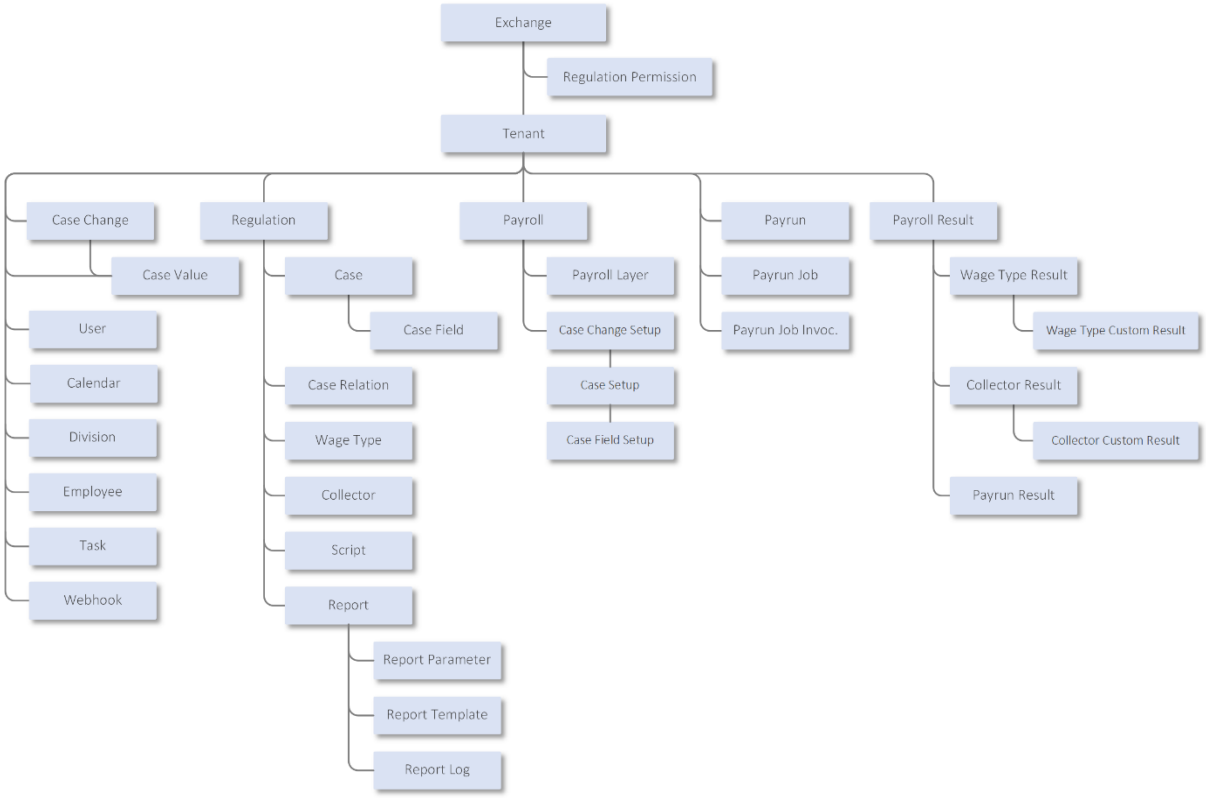
- Migrations and integrations (e.g., interfaces)
- for the administration of regulations and Clients
- the testing of regulations
- the development of scripts



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

7.1.1 Exchange Model

The Exchange model contains all API objects of the Payroll Engine for data exchange:

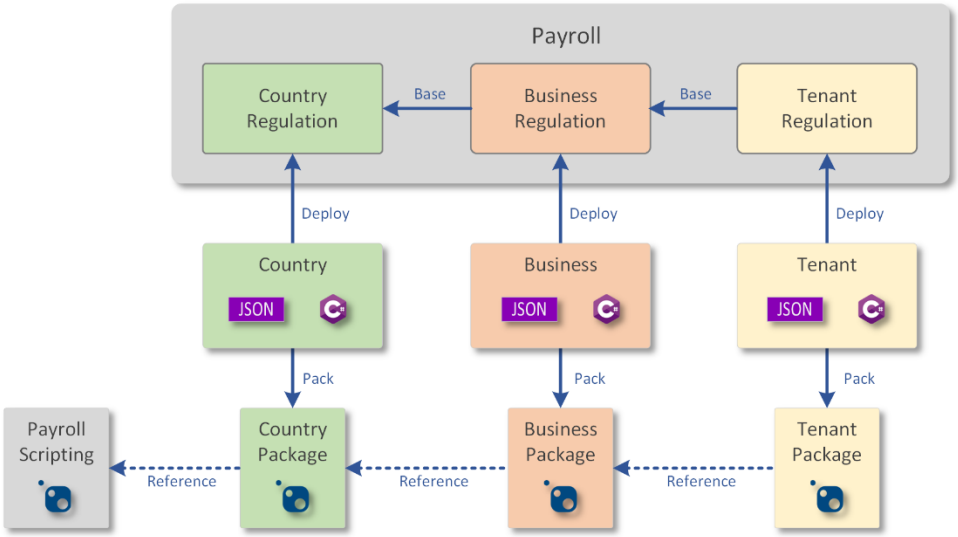


Various tools are available for processing the model (e.g., import and export), which also work for reduced model areas. In addition, there are classes that take over the sequence of model processing and allow the user direct object access.

7.1.2 Regulations Development and Deployment

A regulation is described in JSON files and contains the programmed wage logic as C# code. This is determined as an embedded object expression or in a source code file (.cs). The outsourcing of program logic to source code files is assumed 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 regulations packages, as well as the provision of regulations for the payrun system:



8 Clients

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

- Payroll Console: Import, Export and Test payrolls
- Web application: visual management of the Payroll Engine

8.1 Payroll Console

The Payroll Console application provides various client services functions in one console:

- Payroll Import (JSON)
- 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

8.2 Web App

The web app provides various functions of the client services in a web application:

- Administration of clients and employees
- Collection of case data
- Starting payruns
- Evaluation of the payrun data
- Management of user tasks and developer logs

9 More information

Payroll Engine Website
https://payrollengine.org/

9.1 Open-Source

The Payroll Engine is an Open-Source project hosted on GitHub:
https://github.com/Payroll-Engine