MFSA MALTA FINANCIAL SERVICES AUTHORITY

MFSA: JSON Schema Handbook for Technical Users

V1.0

Contents

Tables

1 Introduction

This document provides technical guidance for using the Harmonised Regulatory Reporting Framework (HRRF) JSON Schemas. It explains the structure of the published schemas, the validation rules they enforce, the dependency expressions supported by the platform, and the requirements for constructing valid JSON submissions.

The HRRF JSON Schemas represent the authoritative specification for regulatory data submissions. They define the expected structure, data types, constraints, and business rules that submitted data must satisfy before it can be accepted by the platform.

This guide is intended to help licence holders understand how to interpret the released schema files, prepare compliant submissions, and integrate HRRF reporting requirements into their reporting and technology solutions. It should be used alongside the published schema artefacts and any accompanying implementation guidance.

1.1 Document Information

This document provides guidance on the structure and organisation of JSON schema files, including the data types, formats, validation rules, and dependency expressions used to define regulatory reporting requirements. It explains how conditional validation logic is represented within the schemas and outlines the principles for constructing valid JSON submissions. The document also provides implementation guidance and recommended practices to help reporting entities, technical implementers, and system integrators develop, validate, and submit compliant regulatory reporting data using the published HRRF schema artefacts.

Table 1.1. Summary document information.

Document Item	**Value**
Audience	Technical implementers, data providers, reporting teams, quality reviewers, and system integrators.
Scope	Public description of HRRF JSON schema files and guide to construct valid JSON submissions.
Release status	Public technical documentation draft.
Last updated	10/07/2026
Internal implementation details	Excluded as the released schema files are the public contract.

Note: Should there be any discrepancy between this document and the published JSON schema files, the JSON schema files take precedence. The purpose of this document is to serve as a guide to explain and support the schema implementations.

2 Executive Summary

The HRRF JSON Schema package defines the structure, validation rules, metadata, and conditional logic used to create and validate HRRF JSON submissions. Each schema specifies the datapoints that may be reported, the expected data type for each datapoint, applicable validation constraints (such as enumerations, ranges, formats, and text patterns), and any dependency rules that determine when additional datapoints are required because of another.

This document serves as a public technical specification for organisations implementing HRRF reporting requirements. Its purpose is to explain the structure and usage of the published schema artefacts so that reporting entities, technical implementers, data providers, and system integrators can construct valid submissions using the schema definitions alone. The document focuses exclusively on externally visible schema components and validation behaviour. Internal database structures, generation processes, processing pipelines, staging environments, and other implementation-specific details are intentionally excluded.

A key feature of the HRRF JSON schema framework is its support for conditional reporting requirements through custom dependency rules. These rules allow datapoints to become applicable only when specific conditions are met, enabling complex business requirements to be represented within a machine-readable validation framework.

Important: The custom dependency keyword is intentionally defined as *requiredIfConditionRefernce* within the published schema artefacts. This is required for compatibility with the validation engine. Care must be taken to use this exact keyword as it appears in the published schemas.

The table below summarises key concepts referenced throughout this document.

Table 2.1. Key document concepts.

Concept	Meaning
Schema	A machine-readable contract that defines valid datapoints, data types, validation constraints, and conditional requirements.
Submission	A JSON object containing reported datapoint values. Each property represents a versioned datapoint identifier, such as FINS_PR_2641_v1.
Module Schema	A schema containing the datapoints associated with a single reporting module, such as GI, PR, AML, SG, OP, or any other module abbreviation.

ALL Schema	A consolidated schema that combines all modules within a reporting package. This schema is typically used to validate complete submissions.
Dependency Rule	A conditional validation rule that evaluates one or more datapoints and determines whether additional datapoints become required or applicable.

3 Schema Package Structure

The HRRF schema package is organised using a hierarchical structure that enables reporting requirements to evolve over time while maintaining version control and traceability. Schemas are grouped by licence code and submission version, allowing reporting packages to be managed independently and updated as regulatory requirements change.

A submission version represents a complete reporting package that defines the set of module schemas applicable for a particular reporting cycle or release. Within a submission version, some modules may be newly introduced or amended, while others may remain unchanged from previous releases. This approach allows changes to be implemented at a module level without requiring the entire reporting package to be redesigned.

3.1 Package Hierarchy

The schema package follows the general structure shown below:

```
schemas/
  <LicenceCode>/
    <SubmissionVersion>/
      TAX_<Licence>_<Module>_<Period>_<Version>_<Licence>_<Module>.json
      TAX_<Licence>_ALL_<Period>_<Version>_<Licence>_ALL.json
```

Each submission package typically contains one or more schema artefacts, with each artefact serving a specific purpose within the reporting framework.

3.1.1 Licence Code

A licence code uniquely identifies the reporting population, regulatory framework, or reporting regime for which a schema package has been created. It acts as the highest-level grouping within the schema package structure and determines which reporting modules, datapoints, and validation rules apply to a particular reporting obligation.

Each licence code is associated with one or more submission versions, allowing reporting requirements to evolve over time while maintaining a clear separation between different reporting populations and regulatory frameworks.

The licence code forms part of the schema file naming convention and directory structure, ensuring that schema artefacts can be easily identified, versioned, and managed.

The licence code is used to group related schema artefacts into reporting packages and identify the applicable reporting framework or reporting population. It also supports version management across reporting releases whilst enabling multiple reporting frameworks to coexist within the HRRF platform.

3.1.2 Submission Version

A submission version represents a specific release of a reporting package for a given licence code.

Each submission version defines the complete collection of module schemas that are applicable for that reporting cycle. New versions may introduce additional datapoints, modify validation rules, add new modules, or retire existing requirements while preserving compatibility with previously released packages.

More than one submission version may exist for a licence code, allowing reporting requirements to be managed and released in a controlled manner.

3.1.3 Licence

A licence represents the regulated activity, reporting population, or reporting framework for which a schema package has been created. It defines the scope of reporting requirements and determines which modules, datapoints, validation rules, and dependency logic apply to a submission.

3.1.4 Module Schemas

A module schema contains the datapoints and validation rules associated with a single reporting module defined within the reporting package.

Module schemas are primarily intended for organisations that would like to validate, generate, or process individual reporting modules independently. All datapoint definitions, constraints, metadata, and module-specific dependency logic are contained within the module schema.

Where supported by the release package, dependency rules may be defined directly at property level within the module schema.

3.1.5 ALL Schema

The ALL Schema combines all module schemas belonging to a particular licence and submission version into a single consolidated validation artefact.

This schema is typically used when constructing a complete submission and for validating cross-module dependency logic if applicable. It is also used for performing end-to-end validation across all reporting modules and for testing reporting integrations prior to submission.

In addition to containing the datapoints from all modules, the ALL Schema centralises dependency rules and conditional requirements. These dependency definitions are typically extracted and maintained as dedicated schema elements at the top level of the schema to simplify validation and improve readability.

The ALL Schema should be considered the primary validation artefact for complete submissions, as it contains the full set of datapoints and dependency rules required to validate an entire reporting package.

3.1.6 Dependency-Only Schema

In some releases, an additional dependency-only schema may be distributed. This artefact contains only the dependency rules extracted from the reporting package and may be used by advanced implementations that wish to process dependency logic separately from structural validation. The dependency-only schema is optional and is not required when dependency rules are already included within the ALL Schema.

4 High-Level JSON Schema Anatomy

Every HRRF schema file follows a consistent structure based on the JSON Schema Draft-07 specification, with a small number of HRRF-specific extensions used to support conditional validation and dependency management.

Understanding the key components of a schema will help implementers interpret validation rules, construct compliant submissions, and troubleshoot validation issues.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "https://.../<schema-file>.json",
  "title": "HRRF Taxonomy Schema - ...",
  "type": "object",
  "properties": {
    "FINS_PR_2641_v1": {
      "type": "integer",
      "minimum": 0
    },
    "FINS_PR_2648_v1": {
      "type": "integer",
      "maximum": 100
    },
    "FINS_GI_11_v1": {
      "type": "string",
      "enum": [
        "Yes",
        "No",
      ]
    }
  },
  "required": [
    "FINS_GI_11_v1"
  ],
  "additionalProperties": false,
  "requiredIfConditionRefernce": [
    "[{[FINS_PR_2641_v1, FINS_PR_2648_v1];(${FINS_GI_11_v1} <> 'Yes')}]"
  ]
}
```

The following attributes provide information about the schema itself.

Table 4.1. Schema member properties.

Schema Member	Standard?	Purpose
$schema	Yes	This property identifies the version of JSON Schema specification used by the schema. HRRF schema is based on Draft-07. This enables validation tools and applications to interpret the schema using the correct JSON Schema standard.
$id	Yes	This provides a unique identifier for the scheme artefact. The value should be

		treated as a globally unique schema reference and may be used by validation tools when resolving schema references.
title	Yes	The title provides a human-readable description of the schema. The title is intended for documentation and identification purposes and does not influence validation behaviour.
type	Yes	The type defines the root structure of the submission. All HRRF submissions are represented as JSON objects containing datapoint identifiers and their associated values.
properties	Yes	The properties attribute defines the complete set of datapoints that may appear within a submission. Each property represents a versioned datapoint identifier and contains the validation rules applicable to that datapoint.
required	Yes	This property defines which datapoints must always be present in a submission. Any datapoint listed within the required array must be included regardless of any dependency conditions. If a required datapoint is omitted, validation will fail.
additionalProperties	Yes	This property controls whether datapoints not defined in the schema are permitted. When this is set to “false” then only datapoints defined under properties are allowed. Any non-defined datapoints will cause the validation to fail.
requiredIfConditionRefernce	No	This attribute is an HRRF-specific extension used to represent conditional reporting requirements. These conditions only apply when specific business conditions are satisfied.

5 Datapoint Property Anatomy

Each datapoint defined within the properties section contains a collection of attributes that describe the datapoint's purpose, expected value, validation requirements and contextual metadata. Together, these attributes provide both the technical validation rules required to validation submissions and the business context needed to present datapoints in user interfaces, forms, templates and reporting tools.

```
"FINS_PR_2641_v1": {
  "title": "Total Amount",
  "description": "Enter the total reported amount.",
  "type": "number",
  "propertyName": "FINS_PR_2641_v1",
  "section": "Prudential Return",
  "sequence": 2641,
  "table_header": "Own funds requirement",
  "row_value": "Total exposure amount",
  "examples": [
    1000
  ]
}
```

Table 5.1. Properties components defined.

Property keyword	Standard?	Purpose
title	Yes	The title provides a short, human-readable label for the datapoint.
description	Yes	The description provides additional guidance explaining the information that should be reported.
type	Yes	This defines the expected JSON data type for the datapoint.
propertyName	Yes	This contains the versioned datapoint identifier.
section	No	This identifies the reporting section to which the datapoint belongs.
sequence	No	The sequence defines the display order of a datapoint. These should typically be presented in ascending sequence order.
table_header	No	This provides table context for datapoints that originate from a tabular reporting structure.
row_value	No	This provides row-level context for datapoints that originate from matrix, tabular or repeating structures.
examples	Yes	This provides illustrative example values which are intended to assist in understanding the expect format and content of a datapoint. These do not form part of the validation logic.

5.1 Datapoint Identifiers and Versioning

Every reportable datapoint with an HRRF schema is represented by a versioned identifier. This identifier serves as the unique reference for the datapoint and is used both as the JSON property key in submissions and as the property name inside the schema.

To aid interpretation and schema consumption, the same identifier is also included in the propertyName metadata attribute, providing a consistent and self-describing reference regardless of how the schema is processed or displayed.

A typical datapoint identifier follows the following structured naming convention:

```
FINS_PR_2641_v1
|    |  |    |
|    |  |    +-- version suffix
|    |  +------- datapoint
|    +---------- module abbreviation
+--------------- licence
```

Table 5.2. Datapoint identifier components.

Identifier part	Example	Description
Licence	FINS	Identifies the taxonomy, licence family or reporting population for which the datapoint is applicable.
Module abbreviation	PR	Identifies the reporting module containing the datapoint.
Datapoint	2641	Underlying datapoint identifier within the module.
Version suffix	v1	Indicates the version of the datapoint applicable to the current submission package.

Note: Versioned identifiers form part of the public contract between the schema and the submission. Property names used in submitted JSON payloads should always match the identifiers published in the schema exactly, including the version suffix.

5.2 Data Types and Validation Constraints

Every datapoint in an HRRF schema is assigned a JSON data type and may also contain one or more validation rules that restrict the values that can be submitted. Together, these definitions ensure that submitted data is structurally correct, consistent, and aligned with the reporting requirements defined by the schema.

Validation rules may apply to numeric values, text values, arrays, or controlled lists of values. These rules are evaluated during schema validation and must be satisfied for a submission to be accepted.

The type property defines the expected JSON data type for a datapoint.

Table 5.3. Supported data types.

JSON Schema Type	Expected JSON Value	Example
string	Text value	"Yes"
number	Integer or decimal number	123.45
integer	Whole number only	42
boolean	true or false	true
array	Ordered list of values	["A", "B"]
object	Nested JSON object	{"code": "X"}

5.2.1 String Constraints

String constraints allow schemas to control the length, format and structure of text values. These rules work together to ensure that the submitted values are not only text-based but also conform to specific business and technical requirements.

The example below demonstrates several commonly used string validation keywords within a single schema definition for illustration purposes.

```
{
  "type": "string",
  "minLength": 3,
  "maxLength": 10,
  "pattern": "^[A-Z]{2}[0-9]{10}$",
  "format": "email"
}
```

Note: Although multiple string validation keywords can be defined together, they must be logically compatible. For example, a value cannot realistically satisfy both the pattern ^[A-Z]{2}[0-9]{10}$ (a two-letter code followed by ten digits) and format = email simultaneously. The above example is intended to illustrate the available syntax rather than represent a practical example.

Table 5.4. String constraint keywords.

Keyword	Description	Example
minLength	Minimum number of characters.	3
maxLength	Maximum number of characters.	10

pattern	Regular expression that the value must match.	"^[A-Z]{2}[0-9]{10}$"
format	Semantic string format recognised by JSON schema.	See Section 5.2.1.1 below for further examples.

5.2.1.1 String Formats

In addition to defining a datapoint type as string, a schema may specify a format to indicate that the value should follow a recognised structure or convention. Formats provide additional semantic meaning and enable validators to perform more specialised checks beyond basic text validation.

Table 5.5. String value formats.

Format	Description	Example
date	ISO date	"2026-07-01"
date-time	ISO timestamp	"2026-07-01T22:00:00Z"
time	Time value	"22:00:00"
email	Email address	"mfsa@mfsa.mt"
uri	Web address or URI	"https://www.mfsa.mt"

Note: A value such as www.mfsa.mt is not a valid URI under JSON Schema format uri because it has no scheme. Use "https://www.mfsa.mt" or "http://www.mfsa.mt" instead.

5.2.1.2 Enumeration Formats

Enumerations restrict a datapoint to a predefined list of allowed values. The schema used the enum keyword to represent these values as shown in the example below. The generated schema always emits enumerations as JSON arrays, even when the source definition was comma-delimited or pipe-delimited as shown below.

```
{
  "type": "string",
  "enum": [
    "Yes",
    "No"
  ]
}
```

5.2.2 Numeric Constraints

Numeric constraints apply to datapoints with type integer or number. These constraints define acceptable ranges and precision requirements.

The example below demonstrates several commonly used numeric validation keywords within a single schema definition for illustration purposes.

```
{
  "type": "number",
  "minimum": 0,
  "maximum": 100,
  "exclusiveMinimum": 0,
  "exclusiveMaximum": 100,
  "multipleOf": 0.01
}
```

Table 5.6. Numeric constraints keywords.

Keyword	Description	Example
minimum	Inclusive lower bound. Value must be greater than or equal to this number.	0
maximum	Inclusive upper bound. Value must be less than or equal to this number.	100
exclusiveMinimum	Exclusive lower bound. Value must be strictly greater than this number.	0
exclusiveMaximum	Exclusive upper bound. Value must be strictly less than this number.	100
multipleOf	Value must be a multiple of the specified number. (Only applicable to type = number.)	0.01

5.2.2.1 Numeric Formats

In addition to defining a datapoint type as either number or integer, a schema may specify a format to indicate that the value should follow a recognised structure or convention. Formats provide additional semantic meaning and enable validators to perform more specialised checks beyond basic numeric validation.

Table 5.7. Numeric value formats.

Data Format Type	Description	Example
currency	A monetary value expressed in a specific currency.	25000, 150000.25
percentage	A value expressed as a percentage.	100, 25.5

5.2.3 Array Constraints

When a datapoint is defined as an array, the submitted value must be a JSON array. In addition to controlling the array itself, schemas may define rules for the individual items contained within that array.

The example below demonstrates several commonly used array validation keywords within a single schema definition.

```
{
  "type": "array",
  "items": {
    "type": "string",
    "format": "email",
    "enum": [
      "mfsa@mfsa.mt",
      "info@mfsa.mt",
      "comms@mfsa.mt"
    ]
  },
  "minItems": 1,
  "maxItems": 3,
  "uniqueItems": true
}
```

Table 5.8. Array constraints keywords.

Keyword	Description	Example
items.type	Data type expected for each array item.	string
items.format	Optional format applied to each string item.	email
items.enum	Allowed values for each item.	["mfsa@mfsa.mt", "info@mfsa.mt", "comms@mfsa.mt"]
minItems	Minimum number of items in the array.	1
maxItems	Maximum number of items in the array.	3
uniqueItems	When true, duplicate array values are not allowed.	true

5.3 Required Fields

The required keyword defines the datapoints that must always be included in a submission. Any datapoint listed in the required array is considered mandatory and must be present regardless of any dependency conditions or business rules.

Unlike conditional requirements defined through requiredIfConditionRefernce (explained in Section 5.5 below), datapoints contained within the required array are always evaluated and must always be provided.

In the example below, both datapoints FINS_GI_11_v1 and FINS_PR_100_v1 are mandatory datapoints and must be included in every submission governed by the schema. Failing to do so will result in an invalid submission.

```
{
  "required": [
    "FINS_GI_11_v1",
    "FINS_PR_100_v1"
}
```

When a datapoint is not included in the required array, it is considered optional unless another validation rules makes it mandatory, for example through a dependency rule in requiredIfConditionRefernce.

Note: The required array defines mandatory datapoints for all submissions, whereas requiredIfConditionRefernce defines datapoints that become mandatory only under certain conditions.

5.4 Additional Properties Field

The additionalProperties keyword controls whether a submission may contain datapoint identifiers that are not explicitly defined in the schema.

By default, JSON objects can contain any properties. However, HRRF schemas use additionalProperties to restrict submissions to the set of datapoints defined under the schema's properties section.

```
{
  "additionalProperties": false
}
```

When additionalProperties is set to false, the schema operates as a closed validation model, meaning that only datapoints explicitly defined in the schema are permitted. This helps to ensure that submissions only contain recognised datapoints, typographically errors in datapoint identifiers are detected and deprecated or unsupported datapoints are not submitted.

Note: In HHRF schemas, additionalProperties is typically set to false, meaning that every datapoint included in a submission must be defined within the published schema.

5.5 Conditional Dependency Fields

In addition to standard JSON Schema validation, HRRF schemas support conditional reporting requirements through a custom validation keyword named requiredIfConditionRefernce.

This keyword allows the schema to express business rules where certain datapoints only become applicable or mandatory when specific conditions are met. These conditions may depend on the value of another datapoint, the result of a comparison, a combination of multiple conditions, or a calculated expression.

Unlike standard JSON Schema keywords such as required, enum, or minimum, requiredIfConditionRefernce is an HRRF-specific extension and is interpreted by the HRRF dependency validation engine.

A dependency rule consists of two components:

1. A list of datapoints that become required when the condition is satisfied.
2. A condition that is evaluated against the submission data.

5.5.1 Dependency Logic

A typical dependency rule follows the following structure:

```
{
  "requiredIfConditionRefernce": [
    "[{[FINS_PR_2641_v1, FINS_PR_2648_v1];(${FINS_GI_11_v1} <> 'Yes')}]"
     | |                                 |||+--datapoint---+         | |
     | |                                 ||+----condition------------+ |
     | |                                 |+--------separator           |
     | +---- target list ----------------+                           |
     +--------- wrapper ---------------------------------------------+
  ]
}
```

The above dependency rule example means that when the datapoint FINS_GI_11_v1 is not equal to "Yes" then the other datapoints FINS_PR_2641_v1 and FINS_GI_11_v1 must also be reported.

Table 5.9. Conditional reference components.

Component	Example	Description
Wrapper	[{ ... }]	Container format expected by the custom validator.
Target list	[FINS_PR_2641_v1, FINS_PR_2648_v1]	Datapoints that become required/evaluated when the condition is true.

Component	Example	Description
Separator	;	Separates the target list from the condition.
Condition	(${FINS_GI_11_v1} <> 'Yes')	Boolean expression evaluated against the submitted values.
Datapoint reference	${FINS_GI_11_v1}	Reference to another datapoint value in the same submission.

5.5.2 Conditional Logic

Dependency rules are often defined using business-friendly expressions that are easier for subject matter experts to author and review. Before being published within a schema, these expressions are transformed into the syntax required by the HRRF dependency validator.

During schema generation, the engine converts dependency expressions into validator-compatible syntax and resolves datapoint references to the correct versioned identifiers published within the schema. This ensures that dependency rules remain both human-readable and machine-evaluable.

The table below illustrates how dependency expressions are interpreted and transformed by the schema generation engine

Table 5.10. Expression translations.

Source Expression	Generated Expression
FINS_PR_2641 > 0	${FINS_PR_2641_v1} > 0
FINS_GI_11 = "Yes"	${FINS_GI_11_v1} = 'Yes'
SUM(A,B,C) > 0	(${A_v1} + ${B_v1} + ${C_v1}) > 0
AND(A = "Yes", B <> "No")	${A_v1} = 'Yes' AND ${B_v1} <> 'No'
OR(A = "Yes", B = "Yes")	${A_v1} = 'Yes' OR ${B_v1} = 'Yes'

Dependency definitions may originate from base datapoint identifiers that do not contain a version suffix. The schema generation process automatically resolves the base identifiers to the corresponding datapoint applicable to the selected submission package.

Note: Always use the versioned identifiers published within the schema and do not attempt to derive version suffixes independently.

5.5.2.1 Boolean Logic

Many dependency rules require multiple conditions to be evaluated together. The HRRF dependency validator supports Boolean logic with AND, OR and NOT operators. These operators allow complex business rules to be represented while remaining readable and transparent.

Table 5.11. Boolean logic.

Operator	Description	Example
AND	Both conditions must be satisfied.	AND(A = "Yes", B <> "No")
OR	At least one condition must be satisfied.	OR(A = "Yes", B = "Yes")
NOT	The reverse condition must be satisfied.	NOT(A="Yes")

Boolean operators can be combined to represent more complex business rules as can be seen below.

Table 5.12. Complex Boolean logic.

Example	Description
AND(OR(A = "Yes", B = "Yes"), C <> "No")	At least one of A = "Yes" or B = "Yes" must be satisfied and C <> "No" must always be satisfied.
OR(A = "Yes", AND(B = "Yes", C <> "No"))	At least one of the following two conditions must be satisfied: 1. A = "Yes" must be satisfied. 2. Both B = "Yes" and C <> "No" must be satisfied.

Note: When multiple operators are combined, parentheses are added to ensure that the intended order of evaluation is maintained. Without them, the meaning of the condition could be interpreted differently by the validation engine.

5.5.2.2 Supported Operators

Dependency conditions are evaluated using a set of supported operators that allow values to be compared, tested against lists, or checked for presence. These operators form the core components of dependency expressions and can be combined with the Boolean logic operators defined above to create more complex reporting rules.

The table below summarises the operators supported by the HRRF dependency validator.

Table 5.13. Supported operators.

Operator	Description	Compatible data types	Example
=	Equal to	String, number, integer, boolean, or null depending on field type.	A = "Yes"
<>	Not equal to	String, number, integer, boolean, or null depending on field type.	B <> "Yes"
>	Greater than	Number or integer.	A > 10
<	Less than	Number or integer.	B < 10
>=	Greater than or equal to	Number or integer.	A >= 10
<=	Less than or equal to	Number or integer.	B <= 10
IN (...)	Value is one of a list	Typically string or enumerated fields.	A IN ("Yes", "No")
NOT IN (...)	Value is not one of a list	Typically string or enumerated fields.	NOT IN ("Yes")
IS NULL	Value is absent or null	Any field type.	A IS NULL
IS NOT NULL	Value is present and non-null.	Amy field type.	B IS NOT NULL

Note: Supported operators enable dependency rules to evaluate values, thresholds, lists and data presence. When combined with Boolean logic operators, they provide a flexible mechanism for expressing complex conditional reporting requirements within HRRF schemas.

5.5.2.3 SUM Expressions

Dependency conditions may contain arithmetic calculations that evaluate multiple datapoints together. The most common example is the use of the SUM(...) function to calculate a total before applying a comparison or threshold condition.

Because the HRRF dependency validator evaluates arithmetic expressions directly, any spreadsheet-style SUM(...) function is converted into an equivalent arithmetic addition expression during schema generation.

Table 5.14. SUM operator.

Operator	Description	Example
SUM (...)	Evaluates the total of values.	SUM(A,B,C)

Similarly, SUM operator can be used in combination with other operators as shown below.

Table 5.15. SUM operator in conjunction with other logical operators.

Example	Description
SUM(A, B, C) > 0	The total of A, B and C must be greater than zero.
SUM(A, B) > SUM(C, D)	The total of A and B must be greater than the total of C and D.
AND(SUM(A, B) > 0, C = "Yes")	Both conditions below must be satisfied: 1. The total of A and B must be greater than zero. 2. C = "Yes" must be satisfied.

5.5.2.4 NULL Handling

Dependency conditions may reference datapoints that are not present in a submission or that contain a null value. The way these situations are evaluated is referred to as null handling.

In HRRF schemas, null handling is primarily controlled by the custom dependency validation engine and by the contents of the submitted JSON payload. The published schema does not normally introduce additional null-check logic unless such checks form part of the original dependency definition.

By default, the schema generation process does not automatically inject null guards into dependency expressions. For example, the following condition: *${FINS_GI_11_v1} = 'Yes'* is not automatically converted to *${FINS_GI_11_v1} = 'Yes' AND ${FINS_GI_11_v1} IS NOT NULL* unless this logic was explicitly authored as part of the original dependency rule.

Note: HRRF schemas do not automatically add null-handling conditions during schema generation. Where dependency logic depends on the presence or absence of a value, null checks should be explicitly defined using *IS NULL* or *IS NOT NULL* within the dependency expression.

5.5.3 Placement of Dependency Rules

In published HRRF schema packages, dependency rules may be stored in one of two locations depending on the schema type.

5.5.3.1 ALL Schema

The recommended ALL Schema consolidates dependency rules at schema level.

```
{
  "properties": { … },
  "required": [ … ],
  "additionalProperties": false,
  "requiredIfConditionRefernce": [ … ]
}
```

This approach improves readability of individual datapoint definitions and centralises dependency management. It also simplifies processing and validation while supporting cross-module dependency evaluation. Hence, the ALL Schema is the recommended artefact for full-submission validation.

5.5.3.2 Module Schemas

Some release packages may include dependencies individual datapoint definitions.

```
{
  "properties": {
    "FINS_PR_2641_v1": {
       …
       "requiredIfConditionRefernce": [ … ]
     }
  }
}
```

This layout is valid but is generally intended for module-level validation scenarios.

6 Submission Construction from the Schema Alone

A valid HRRF JSON submission can be constructed directly from the published schema without requiring access to any internal implementation details. The schema contains all information necessary to identify valid datapoints, determine which values are expected, apply validation rules, and evaluate conditional reporting requirements.

This approach can be used for manual preparation of submissions, dynamic form and user interface generation, automated template creation, data validation and quality assurance processes, system integrations and submission engines and AI/LLM-assisted submission generation.

The process below outlines the recommended approach for constructing a valid submission from the schema.

1. Select appropriate schema
    a. Select the schema package that corresponds to the required licence code and submission version.
    b. For complete submissions, use the published ALL Schema, as it contains the full set of datapoints, validation rules, and dependency definitions applicable to the reporting package.
2. Identify available datapoints
    a. Review the schema's properties section.
    b. Each property represents a valid datapoint that may be included in a submission.
    c. The property name acts as the JSON key that must be used within the submission payload.
    d. Datapoint identifiers should be copied exactly as defined in the schema, including any version suffixes.
3. Organise datapoints for data collection
    a. Use the contextual metadata provided within each datapoint definition to organise information logically.
    b. Relevant metadata includes:
        i. section: Groups related datapoints together.
        ii. sequence: Defines display order.
        iii. table_header: Provides column context.
        iv. row_value: Provides row context.
    c. This metadata can be used to reconstruct the original reporting structure and improve the user experience during data collection.
4. Determine Expected Values
    a. Review the validation rules associated with each datapoint to determine the expected JSON value and any applicable restrictions.

    b. Validation rules may include:
        i. Data type (type)
        ii. Format (format)
        iii. Enumerations (enum)
        iv. Constants (const)
        v. Numeric constraints
        vi. String constraints
        vii. Array rules
    c. These properties collectively define the values that may be reported for a datapoint.
5. Populate Mandatory Datapoints
    a. Review the schema's required array.
    b. All datapoints listed in required must be included in the submission.
    c. Each required datapoint must contain a value that satisfies the applicable validation rules.
    d. Failure to provide any required datapoint will result in validation failure.
6. Evaluate Conditional Requirements
    a. Review the schema's requiredIfConditionRefernce rules.
    b. All dependency rules should be evaluated against the current submission values to determine whether additional datapoints must be populated.
    c. When a dependency condition evaluates to true, the datapoints identified in the target list become mandatory and must be populated.
    d. A datapoint that does not appear in the required array may still become required through a dependency rule.
7. Exclude Undefined Datapoints
    a. Review the additionalProperties setting.
    b. When this value is set to false, only datapoints explicitly defined in the schema may appear in the submission.
    c. Do not include:
        i. Internal fields
        ii. Temporary fields
        iii. Derived fields
        iv. User-defined properties
        v. Datapoints from other schema versions
8. Validate the Completed Submission
    a. Once all datapoints have been populated:
        i. Validate the JSON structure against the schema.
        ii. Validate all dependency conditions.
        iii. Verify that all mandatory datapoints have been populated.

iv. Verify that all conditionally required datapoints have been provided.
v. Confirm that all values satisfy their respective validation rules.
vi. Ensure that no undefined datapoints are included.

b. A submission should pass:
    i. Standard JSON Schema validation
    ii. HRRF dependency validation
c. Only submissions that pass both validation layers should be considered ready for submission.

7 Common Implementation Errors

The table below highlights some of the most common issues encountered when constructing or validating HRRF submissions. Understanding these issues and their causes can help reduce validation errors and improve the quality of submitted data.

Table 7.1. Common errors and their resolutions.

Issue	Why It Happens	Resolution
Using unversioned keys in submissions	Datapoints in HRRF schemas are published using versioned identifier. Unversioned identifiers cannot be matched to schema definitions.	Use the exact property key defined in the schema, including the version suffix, for example FINS_PR_2641_v1.
Submitting www.mfsa.mt for format uri	THE JSON Schema uri format required a complete URI, including a scheme such at https:// or http://.	Use https://www.mfsa.mt instead of www.mfsa.mt.
Using values that are not defined in an enum	Enumeration values are validated using exact matching. Values that differ in spelling, case, spacing, or punctuation are considered invalid.	Populate the value exactly as it appears in the enum list.
Including unknown datapoints	The schema typically defines "additionalProperties": false, which prevents the submission of undefined datapoints.	Remove any datapoints that do not appear in the schema properties section.
Ignoring conditional requirements	Some datapoints become mandatory only when a dependency condition evaluates to true.	Evaluate all requiredIfConditionRefernce rules after populating mandatory datapoints.
Treating sequence as JSON order	JSON object ordering is not significant for validation purposes. The sequence attribute exists only as metadata.	Use sequence to control display order and presentation, not JSON construction order.
Treating examples as allowed values	The examples attribute provides guidance only and does not define allowable values.	Use validation keywords such as enum, const, type and other constraints to determine valid values.

8 Public Artefact Contract

The HRRF schema artefacts are designed to be self-describing. A schema consumer should be able to understand the reporting requirements, generate data collection templates, construct valid submissions, and perform both standard and HRRF-specific validation using only:

- The published schema files; and
- This technical specification.

No knowledge of internal systems, databases, processing pipelines, or implementation-specific logic is required to interpret or use the schema.

The schema therefore acts as the authoritative public contract between reporting entities and the HRRF platform, defining what data can be reported, how it should be structured, what validation rules apply, and under what conditions additional information becomes required.

The schema contains all information necessary to:

- Identify valid datapoints.
- Understand the purpose of each datapoint.
- Present questions in a meaningful order.
- Reconstruct sections, tables, and matrix-style reporting layouts.
- Determine valid values and validation constraints.
- Identify mandatory datapoints.
- Evaluate conditional reporting requirements.
- Validate complete submissions.

The table below summarises where this information can be found within the schema.

Table 8.1. Schema components.

Consumer Need	**Schema Component**
What fields exist?	properties
What are the field labels?	properties.<id>.title
What guidance should users see?	properties.<id>.description
What order should questions appear in?	properties.<id>.section and properties.<id>.sequence
How do I rebuild tables or matrices?	properties.<id>.table_header and properties.<id>.row_value

What values are allowed?	type, format, enum, const, minimum, maximum, pattern, items
Which fields are always required?	required

9 Glossary

Term	Definition
ALL Schema	A consolidated schema containing all reporting modules applicable to a particular licence and submission version.
Array	A JSON data type representing an ordered collection of values.
Boolean	A JSON data type that can only contain the values true or false.
Conditional Dependency	A rule that makes one or more datapoints required when a specified condition is satisfied.
Datapoint	A reportable item of information represented by a unique identifier within the schema.
Dependency Rule	A conditional expression evaluated by the HRRF dependency validator to determine whether additional datapoints become required.
Enum	A fixed list of permitted values that may be submitted for a datapoint.
Format	A semantic constraint applied to a string value, such as email, date, or uri.
HRRF	Harmonised Regulatory Reporting Framework.
JSON	JavaScript Object Notation, a structured data interchange format used for HRRF submissions.
JSON Schema	A standard specification used to define the structure, types, and validation rules of JSON documents.
Licence	A reporting framework, reporting population, or regulated reporting obligation represented within the HRRF platform.
Licence Code	The technical identifier used to represent a licence within schema packages and file names.
Module	A logical grouping of related datapoints within a reporting package.
Module Schema	A schema containing the datapoints and validation rules for a single module.
Object	A JSON data type that contains one or more related properties.
Property	A named element within a JSON object. In HRRF schemas, properties represent datapoints.
Property Key	The versioned datapoint identifier used as a JSON property name.
Required Datapoint	A datapoint listed in the schema's required array that must always be present in a submission.
requiredIfConditionRefernce	HRRF-specific schema keyword used to define conditional dependency rules.

Schema Artefact	A published JSON Schema file used for submission construction and validation.
Section	Metadata that groups related datapoints together for presentation purposes.
Sequence	Metadata used to define the intended display order of datapoints.
Submission	A JSON payload containing datapoint values reported to the HRRF platform.
Submission Version	A specific release of a reporting package associated with a licence code.
Target Datapoint	A datapoint that becomes required when a dependency condition evaluates to true.
Type	The JSON data type assigned to a datapoint, such as string, number, or array.
URI	A Uniform Resource Identifier, typically a fully qualified web address including a scheme such as https://.
Validation Rule	A constraint that determines whether a submitted value is acceptable.
Versioned Identifier	A datapoint identifier that includes a version suffix, for example FINS_PR_2641_v1.

Appendix A. Complete Sample Example

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id":
"https://example.org/hrrf/FINS/SUBVER_FINS_v1/TAX_FINS_ALL_2026_v1_FINS_ALL.json",
  "title": "HRRF Taxonomy Schema - TAX_FINS_ALL_2026_v1 / FINS / ALL",
  "type": "object",
  "properties": {
    "FINS_GI_11_v1": {
      "title": "Currently authorised",
      "description": "Indicate whether the licence holder is currently authorised.",
      "type": "string",
      "propertyName": "FINS_GI_11_v1",
      "section": "General Information",
      "sequence": 1,
      "enum": [
         "Yes",
         "No"
         ]
      },
    "FINS_PR_2641_v1": {
      "title": "Total active clients",
      "description": "Enter the total number of active clients.",
      "type": "integer",
      "propertyName": "FINS_PR_2641_v1",
      "section": "Prudential Return",
      "sequence": 2,
      "minimum": 0
    },
    "FINS_PR_2648_v1": {
      "title": "Subset of active clients",
      "description": "Enter the subset value.",
      "type": "integer",
      "propertyName": "FINS_PR_2648_v1",
      "section": "Prudential Return",
      "sequence": 3,
      "minimum": 0
    }
  }
  "required": ["FINS_GI_11_v1"],
  "additionalProperties": false,
  "requiredIfConditionRefernce": [
    "[{[FINS_PR_2641_v1, FINS_PR_2648_v1];(${FINS_GI_11_v1} <> 'Yes')}]",
    "[{[FINS_PR_2648_v1];(${FINS_PR_2641_v1} > 0)}]"
  ]
}
```

The following examples illustrates how the schema and dependency rules would be evaluated.

Minimum valid submission:

```
{
    "FINS_GI_11_v1": "Yes"
}
```

Conditional dependency triggered:

```
{
    "FINS_GI_11_v1": "No",
    "FINS_PR_2641_v1": 100,
```

```
      "FINS_PR_2648_v1": 50
}
```

Appendix B. Public Implementation Notes

The published schema artefacts constitute the official public contract between reporting entities and the HRRF platform. They define the valid datapoints, data types, validation constraints, dependency rules, and submission structure applicable to a reporting package.

Consumers should use the schema files as the primary source of truth when building reporting solutions, generating submission templates, validating payloads, or implementing integrations.

Note: Consumers should treat the released schema files as the definitive implementation reference. Where any conflict exists between documentation and schema content, the schema takes precedence. Property names, validation constraints, and dependency rules should always be derived from the published schema artefacts rather than from secondary documentation sources.