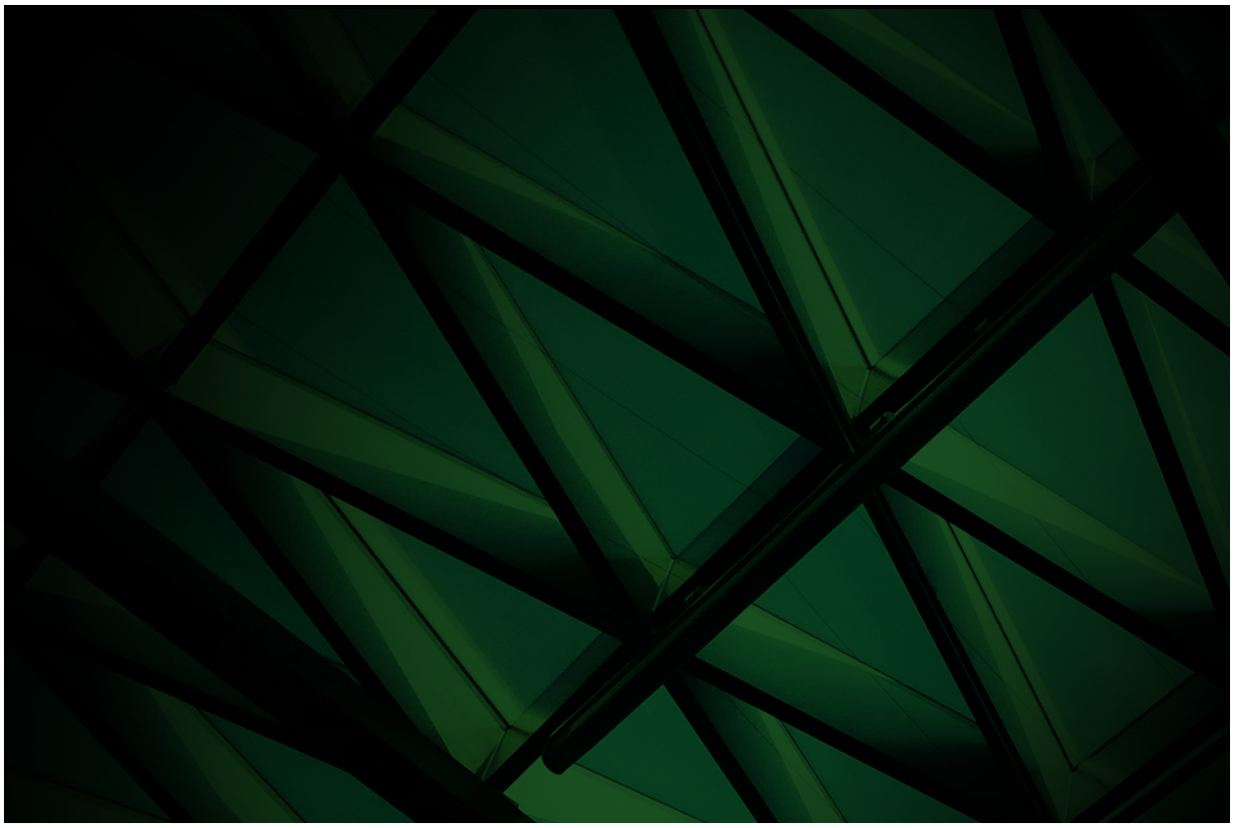


Using the Jade Report Writer White Paper

VERSION 2022

jade platform

Contents

Using the Jade Report Writer

The Jade Report Writer is a generic tool that enables you to configure and design reports for all schemas in your Jade database. It allows flexibility in the format of the data presented in reports. Underlying the formatting of the data, the Jade Report Writer contains a powerful and flexible interface for selecting and accessing the data in your Jade database. This flexibility, however, comes at a cost. As a generic Report Writer, it can never be as efficient as hand-crafted reports.

The purpose of this white paper is to provide you with an understanding of how to use the Jade Report Writer to improve performance during execution of user reports.

It is assumed that you are familiar with the usage of the Jade Report Writer.

Report Execution Overview

The Jade Report Writer is a Jade schema that includes the Report Writer Configuration and the Report Writer Designer applications.

This white paper focuses on the execution performance of your reports; not on performance during report development.

Jade Report Writer reports can be executed from the Report Writer Designer application or from your code. The information in this white paper applies to both execution modes.

Execution of a report under the Jade Report Writer consists of:

1. Creation of a query to select the query input objects required for the report and to extract the selected data fields (properties and methods) from the Jade database.

   Execution of the query created for the report with the Ad Hoc Query Engine (AQE).

2. Creation of the **JadeQueryResult** by the AQE using the Profile Selection Criteria and Sort Order.

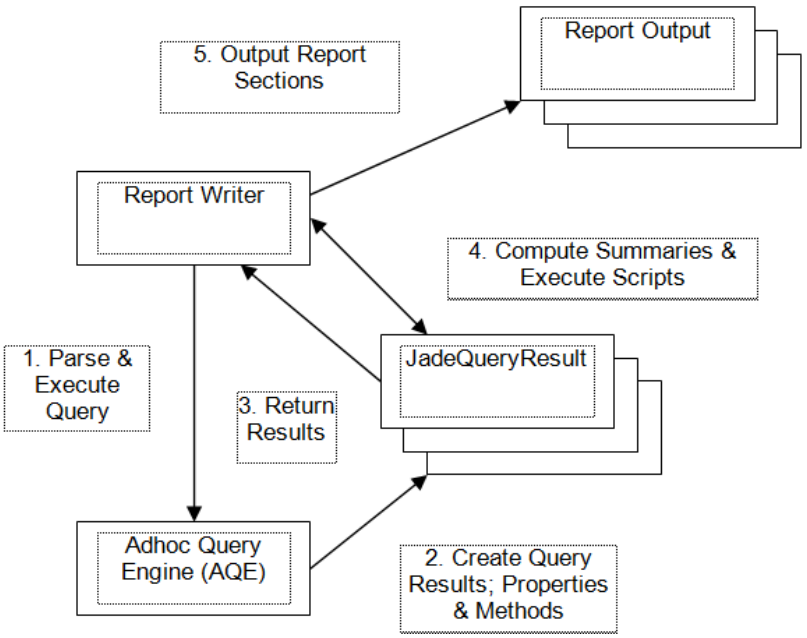
   The **JadeQueryResult** contains the values of the properties and methods used in the report.

3. Return of the **JadeQueryResult** built by the AQE to the Report Writer.

4. Execution of the Report Writer scripts and Report Writer summaries over the **JadeQueryResult**.

5. Output of the formatted report sections.

The following image is fundamental to this white paper.



Query Execution Phase

The query is constructed based on the report collection, the selections from the catalog, the selection criteria from the profile, and the sort order from the profile.

The query is prepared (parsed) and executed in the AQE. All instances of the report collection (or collections, as constrained by the selection criteria, if applicable) are tested for inclusion in the **JadeQueryResult**.

For each instance of the report collection to be included, the AQE reads any dependent objects and retrieves properties and executes methods for these objects, based on your selection from the catalog in the Report Writer Designer application.

The **JadeQueryResult** returned to the Report Writer is a collection of **JadeQueryResultObject** instances - transient objects of a transient class, in the sort order specified, built specifically for this report.

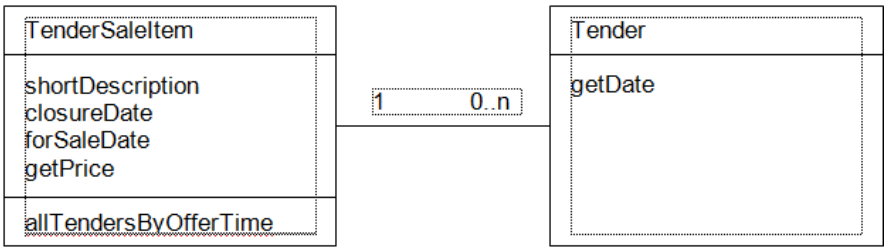
Each **JadeQueryResultObject** contains the values retrieved for a detail line and any values used in scripts or summaries of the report.

The trace output from the AQE can be helpful in understanding and analyzing the efficiency of the generated query.

To display query information in the **jommsg.log** file, set the following parameters in the Jade initialization file.

```
[JadeReportWriter]
QueryPrepTraceOn=true
QueryExecTraceOn=true
QueryStatisticsTraceOn=true
QueryOptimizationTraceOn=true
QueryDataFailureTraceOn=true
```

The following query is from a report over the Erewhon example system. This report uses the **TenderSaleItem** class and its **allTendersByOfferTime** member of **TenderByTimeOfferDict**. This report uses the report collection **TenderSaleItem.instances** as the base collection. For every instance that conforms to the profile selection (**closureDate** between 1 January 2000 and 31 December 2000), it displays the properties **shortDescription**, **closureDate**, and **forSaleDate**, the **getPrice** method, and the **getDate** method for all instances in **allTendersByOfferTime**.



The following query method generates and displays the log.

```
_queryMethod(): JadeQueryResult;
vars
    _x1 : TenderSaleItem;
    _x2 : Tender;
    _queryResult : JadeQueryResult;
begin
    _queryResult :=
    __select
        _x2.getDate           __alias _getDate,
        _x1.shortDescription  __alias _tenderSaleItem_shortDescription,
        _x1.closureDate       __alias _tenderSaleItem_closureDate,
        _x1.forSaleDate       __alias _tenderSaleItem_forSaleDate,
        _x1.getPrice          __alias _tenderSaleItem_getPrice
    __from
        _x1 in TenderSaleItem.instances,
        _x2 in _x1.allTendersByOfferTime
    where
        _x1.closureDate >= '01 January 2000'.Date and
        _x1.closureDate <= '31 December 2000'.Date
    __order __by
        _x1.closureDate __ascending;
    return _queryResult;
end;
```

The query uses a Jade-defined query language based on Object-Oriented SQL (OOSQL), with the following sections.

- Vars: Define each query input object.
  - Each **JadeQueryResultObject** contains values from a **TenderSaleItem** object and a **Tender** object.
- Select: Define all properties and methods whose values are returned in each **JadeQueryResultObject**.

  The alias names are the property names in the transient class created for the **JadeQueryResultObject** for this report.
- From: Define how each query input object is selected.
  - For each instance of **TenderSaleItem**, select each entry of **Tender** in the collection **allTendersByOfferTime**.

- Where: Define selection criteria from the profile selected.
- OrderBy: Define the sort order from the selected profile.

Following the query method, the debug output contains the query plan. This section describes the tasks the AQE needs to execute to find all instances of all objects required for the report output. In this case, the AQE must iterate all instances of **TenderSaleItem** and select the instances that conform to the selection criteria (Task 1). For each of these instances, it then selects each **Tender** class instance in the **allTendersByOfferTime** dictionary (Task 2).

```
JadeAQE: [no tag] --- Query Plan ---
JadeAQE: Task 1 : Iterate 'TenderSaleItem.instances' filter using the criteria
'_x1.closureDate >= '01 January 2000'.Date' (local criteria) and
'_x1.closureDate <= '31 December 2000'.Date' (local criteria)
JadeAQE: Task 2 : Unnest '_x1.allTendersByOfferTime' using the output of Task 1
```

Following the query plan, the debug output contains the Query Execution information. Any report parameters are output and options are displayed, followed by the execution of each task and the number of tuples found for that task. A tuple is a representation for a set of query input objects required for each **JadeQueryResultObject** to be created. In this case, Task 1 (the iterate task) resulted in 31 **TenderSaleItem** objects. Task 2 resulted in 100 <**TenderSaleItem, Tender**> pairs of instances.

The Task Projection phase is when the AQE retrieves, for each of the 100 <**TenderSaleItem, Tender**> **JadeQueryResultObjects**, each selection in the **Select** clause. Each selection is a **getProperty** call or a method call. The result of these calls is stored in a **JadeQueryResultObject**, which is returned to the Jade Report Writer.

```
JadeAQE: [no tag] --- Query Execution - start ---
JadeAQE: No Parameters defined
JadeAQE: Options :
JadeAQE: ReadFailureOption = false
JadeAQE: LockFailureOption = false
JadeAQE: Concurrency Strategy : Repeatable Read Optimistic
JadeAQE: Resource Quota : None
JadeAQE: Task 1 : start
JadeAQE: Task 1 : end 31 - tuples retrieved
JadeAQE: Task 2 : start
JadeAQE: Task 2 : end 100 - tuples retrieved
JadeAQE: Task Projection : start
JadeAQE: Task Projection : end
JadeAQE: [no tag] --- Query Execution - end ---
```

The displayed Query Execution statistics are:

```
JadeAQE: [no tag] --- Query Execution Statistics ---
JadeAQE: NumDbReads :          148
JadeAQE: NumAuthorized :       148
JadeAQE: NumSecurityCalls :      0
JadeAQE: NumOperations :       148
JadeAQE: NumResults :          100
JadeAQE: NumProgressCalls :      1
JadeAQE: NumIfDefineds :         0
JadeAQE: NumLockFailures :       0
JadeAQE: NumReadFailures :       0
JadeAQE: NumTruncations :        0
JadeAQE: ElapsedTime :    00:00:01
```

In the above example, the number of **DbReads** indicates that only 31 of the possible 48 **TenderSaleItems** were within the selection range.

The Execution statistics are defined as:

- **NumDbReads**: The number of objects read.
- **NumAuthorized**: The number of those object reads that were authorized by the **jadeReportWriterCheck** method calls.
- **NumSecurityCalls**: The number of objects read that actually had security methods defined for the class.
- **NumOperations**: The number of internal operations that were performed during the query (for example, reads, join operations), which are a measure of how much work the overall query has done.
- **NumResults**: The number of **JadeQueryResultObjects** returned to the Jade Report Writer. This equates to the number of Detail records for the report.
- **NumProgressCalls**: The number of callbacks to the **ProgressBar** to check for cancellation, if enabled.
- **NumIfDefineds**: The number of exceptions of the type 1086 *(Class of object is invalid in this context)* and 1090 *(Attempted access via null object reference)* raised in this report.

  These exceptions will occur when null references or invalid down-casting occurs during the data access. They do not represent errors in the report or data.

  These exceptions result in null data being recorded in the report output.
- **NumLockFailures** and **NumReadFailures**: The number of lock exceptions and read *(Object Not Found)* exceptions that occurred during the data access.
- **NumTruncations**: The number of **String** or **Binary** truncations that occurred when storing the value into the **JadeQueryResultObject**.

Use of Report Collections

The report collection defined as the root of the report can affect the Jade Report Writer execution time and output.

Where possible, you should select a collection whose keys match the report selection criteria, to limit the number of objects that need to be searched. If you expect that the report should be using a dictionary to optimize the retrieval, you can check the debug output to verify this.

The use of existing collections is preferable to the use of virtual collections (for example, instances). The use of instances requires the collection to be created for the report run, which may have an overhead if a large number of instances exist. In addition, it is inherently unordered and no optimization using search conditions or sort order is possible.

The use of instances is also not recommended, because unexpected results may be obtained. Extraneous data may be tested or included in the report if invalid instances are present in the database. Even if the data does not appear in the report, the reading and evaluation of the data adds unnecessary overhead to the report processing.

In the following example, the use of the root collection **Company.firstInstance** allows the use of the dictionary **Company.firstInstance.allTenderSaleItemsByDate** by the AQE to optimize the retrieval of the required objects by starting the selection at the first key; that is, "01 January 2000". While the result is the same number of objects as the previous example, the use of the dictionary limits the first task to accessing only 31 objects instead of all 48 objects.

```
JadeAQE: Task 1 : Iterate 'Company.firstInstance.allTenderSaleItemsByDate'
optimized retrieval using key(s) 'closureDate, forSaleDate'
filter using the criteria
'_x1.closureDate >= '01 January 2000'.Date' (local criteria) and
'_x1.closureDate <= '31 December 2000'.Date' (local criteria)
```

```
JadeAQE: Task 2 : Unnest '_x1.allTendersByOfferTime' using the output of Task 1
JadeAQE: Task 1 : start
JadeAQE: Task 1 : end 31 - tuples retrieved
JadeAQE: Task 2 : start
JadeAQE: Task 2 : end 100 - tuples retrieved
JadeAQE: NumAuthorized :        131
JadeAQE: NumResults :           100
JadeAQE: NumDbReads :           131
JadeAQE: NumOperations :        231
```

Joins and Cross-Product Reports

Where items from two or more collections from the same object are reported on or multiple root collections are used, you can define a relationship (join) between one or more items in each collection. This may be necessary when the relationship has not been defined in the Jade system; for example, a report can have the following two root collections.

```
X1.firstInstance.allX2s

X3.instances
```

When classes **X2** and **X3** have a type attribute, the following join can be defined.

```
x1.allX2s.type = x3.type
```

Normally, there will be no natural relationship and in this case, a cross-product join is created, where each instance of the first collection is reported with each instance of the other collection. This can produce a large amount of output and is often not the intended result.

Using Profile Selection Criteria

The profile selection criteria should be used to minimize the number of objects in the result set. If a large number of objects needs to be searched for selection in the report output, it is more efficient to include this selection in the profile selection. This limits the number of **JadeQueryResultObjects** passed back from the AQE.

If a report outputs summary information based on selected criteria, it is important that any objects that fit none of the selected criteria are eliminated in the profile selection, rather than through evaluation of the object in the detail section. For example, if the summaries reported on are **sumX1** and **sumX2** where the criteria for selection in each category is **x1** and **x2**, the profile selection criteria should include the following, to eliminate all objects that fit neither summary in the AQE task execution.

```
x1 or x2
```

Where possible, the profile selection criteria should use literals or parameters. Selection using scripts is more flexible but less efficient, as the scripts are evaluated in the report output phase rather than in the query. This means that the AQE cannot eliminate them from the **JadeQueryResult**, and these unused objects will still be included in the **JadeQueryResult** and have all properties and methods on them evaluated prior to their exclusion.

Concurrency Strategy

You can specify a concurrency strategy to be used when accessing objects and collections during report execution. Selection of a concurrency strategy involves a trade-off between efficiency (of both the report execution and any concurrent updates) and correctness and consistency of the report output.

The options are set by calling the **JadeReportWriterReport::setQueryOption** method or by selecting the **Concurrency Strategy** value on the **Options** sheet of the Report Query Options form, accessed by selecting the Report menu **Query Options** command in the Jade Report Writer Designer application.

The concurrency strategy values are as follows.

- None
  - No retained shared locks of collections or objects.
  - The use of automatic cache coherency ensures that non-collection objects in the persistent cache are up to date.

    Collection objects are up to date, because they are shared-locked while accessed.
  - Low impact on concurrent updates.
  - Fastest report execution.
  - Results may be inconsistent if concurrent updating transactions modify objects or collections being reported on.
  - Use when reporting on static (historical) data or when inconsistency in report output due to changing data is acceptable.
- Read Committed
  - Automatic cache coherency ensures that non-collection objects in the persistent cache are up to date.

    If automatic cache coherency is not on, every accessed object is resynchronized.
  - Acquires shared locks on collections iterated during the Query Execution phase.
  - Shared locks are retained for the duration of the Query Execution phase.
  - Impacts on concurrent transactions that update the collections being read in the report.
  - Guarantees collections are not modified while being read.
- Repeatable Read Optimistic
  - Automatic cache coherency ensures that non-collection objects in the persistent cache are up to date.

    If automatic cache coherency is not on, every accessed object is resynchronized.
  - Acquires shared locks on collections iterated during the Query Execution phase.
  - Reads the objects in the collections to check for inclusion in the report, using the selection criteria.
  - Acquires shared locks on objects that satisfy the selection criteria.
  - After the shared lock on the object is acquired, a check is made that the object had not been updated. An exception is raised if an object has changed.
  - Shared locks are retained for the duration of the Query Execution phase.
  - Locks only objects in the collections which are included in the report output.
  - Use when reporting on data that may be changeable, but for which you need a consistent view.

- Repeatable Read Pessimistic
  - Automatic cache coherency ensures that non-collection objects in the persistent cache are up to date.

    If automatic cache coherency is not on, every accessed object is resynchronized.
  - Acquires shared locks on collections iterated during the Query Execution phase.
  - Acquires shared locks on all objects in these collections.
  - Acquires shared locks on all objects referenced in the report output.
  - Shared locks are retained for the duration of the Query Execution phase.
  - Guarantees consistent data and report execution.
  - Impacts on concurrent transactions that update the collections or objects being read in the report.
  - Use when reporting on data that may be changeable and for which you must guarantee the report will complete without data updates or exceptions.

Note This option may be expensive when large numbers of objects are reported on, since the combined lock table entries will consume large amounts of memory.

You should avoid this option, if possible.

The use of the lower-concurrency and higher-consistency options may have a significant impact on online processing. To avoid this impact, the reports may need to be run outside of normal online hours or offloaded to an SDS read-only server.

Options for Limiting Report Output

The Jade Report Writer options to limit the report output for print preview are defined in the **Preview** sheet of the Report Designer form, by specifying the maximum:

- Objects to report; that is, the maximum number of **JadeQueryResultObjects** returned to the Report Writer from the AQE.
- Pages to print.

Limiting the number of **JadeQueryResultObjects**, while useful, can be misleading because the number of objects read to retrieve the resultant **JadeQueryResult** number is usually larger than the number of results. This is a result of how the AQE serializes the tasks required for determining which tuples of objects will be in the **JadeQueryResult**, and in what order.

The limit specified for the **Maximum objects to report** value cannot be used until the last task. For example, with the following task list and the maximum objects set to 50, 89 reads occur because 20 and 19 reads are done in Task 1 and Task 2, respectively.

```
Extent Task 1 : Iterate 'Company.firstInstance.allClients'
Extent Task 2 : Unnest 'x1.allRetailSales' using the output of Task 1
Extent Task 3 : Unnest 'x2.myCompany.allAgents' using the output of Task 2
Extent Task 1 : end 20 - tuples retrieved
Extent Task 2 : end 19 - tuples retrieved
Extent Task 3 : end 50 - tuples retrieved
NumResults :            50
NumDbReads :            89
```

Limiting the number of objects read, in many cases, results in no output being produced because the last task is never even executed.

Limiting the objects read in each task may also result in the partial report being completely different from the final report. This would occur, for example, if the read limit were 100 but the first 100 items selected in the first task were subsequently eliminated in the remaining tasks or would not have been at the start of the report in the final output due to sort order.

Report Output Phase

The report output phase includes the execution of scripts and summaries, and the output to the selected output format. If possible, this phase consists of one pass of the **JadeQueryResult** to compute any scripts and summaries, and to print each detail line (and other sections), as required for the report.

However, for some reports, the report output phase must consist of two passes of the **JadeQueryResult**: the reading pass and the printing pass. The additional pass is required when the report contains:

- Summaries that are not running totals and are not reset after printing
- Scripts used in sort order or selection criteria
- Scripts that are referenced by summary fields

When scripts are used in the sort order or selection criteria, the reading pass of the **JadeQueryResult** will execute the script and create the collection of **JadeQueryResultObjects** to be included in the output. The printing pass then outputs the objects using this collection.

When summaries are not running totals and are not reset after printing, the reading pass of the **JadeQueryResult** computes the final total of this summary (which may be used anywhere in the report output or other scripts).

The printing pass then prints the report, using the computed summary results, as required for the report; for example, when a report contains the following.

- Summary (**GroupTotal**) over the field (reset when the group changes)
- Summary (**SumTotal**) over the field (never reset)
- Group Footer Script (**Percentage**) to compute a percentage to be displayed that uses **GroupTotal** and **SumTotal**
- First pass of the **JadeQueryResult** computes **GroupTotal** (value for each group) and the **SumTotal** value
- Printing pass computes (and prints) each **Percentage** value
- Summary (**GroupTotal**) over the field (reset when group changes)
- Summary (**SumTotal**) over the field (reset never)
- Group Footer Script (**Percentage**) to compute a percentage to be displayed that uses **GroupTotal** and **SumTotal**
- First pass of the **JadeQueryResult** computes **GroupTotal** (value for each group) and the **SumTotal** value
- Printing pass computes (and prints) each **Percentage** value

- 10,000 **JadeQueryResults** = 400,000 executions
- 100,000 **JadeQueryResults** = 4,000,000 executions, and so on

The same logical summarization done in a Jade method that iterates through a collection of 100,000 objects and evaluates each object for inclusion in each of 40 possible summaries requires each object to be read once only, and the summaries totaled in local variables. This is obviously going to be much more efficient.

Using Jade Report Writer Scripts

Report Writer scripts are evaluated during the report output phase. A transient method is created for each script. For every instance of the **JadeQueryResult**, each transient method is executed using the **JadeQueryResult** instance. The result of the script is stored in the **JadeQueryResult** instance.

For scripts that are not printed in the Detail section of the report and that are not referenced in other scripts or summaries (for example, if the script is used only in the Group Footer section), the script is executed only for the **JadeQueryResult** instance that is current for the section in which the script is printed.

Note While the use of Jade Report Writer scripts is a powerful and convenient tool, your system designer must keep in mind that providing a method on the class to retrieve the required data is usually more efficient, because the Jade Report Writer script is called as a transient method and every data access is a callback to retrieve the property from the **JadeQueryResult** object.

Using Jade Report Writer Summaries

Report Writer summaries are evaluated during the report output phase. For every instance of the **JadeQueryResult**, each summary value is updated and the value is stored back in the **JadeQueryResult**.

Summary values can be used at any point in the report. Where this usage is before the value has been fully calculated (for example, using group totals in the report detail), an extra pass of the **JadeQueryResult** is done in the report output phase, to calculate the group totals before they are applied to each instance during the Printing pass.

Tip Use the **Evaluation Phases** command in the Report menu of the Jade Report Writer Designer application to show the phase at which each script and summary in the report is evaluated.

Preparing User Data

Your system designer must prepare a Report Writer view of the data upon which the reports are to be run.

The format and organization of the data in the view can affect the efficiency of the Jade Report Writer.

Selecting Data to be Included in the View

We recommend that you include only selected classes and features in the view.

Your system designer should select only those classes and features that are required for the implementation of the types of reports that you require. He or she should consider including classes specifically designed for report use that flatten and summarize the data to be reported on.

Data that is to be displayed in reports for a class needs to be visible and accessible to the Jade Report Writer. You may need to write methods to:

- Expose the data as a method that has no parameters and returns a primitive type.
- Format or interpret the information for the report; for example, if a type is encoded as an **Integer**.
- Combine some common combinations of displayed data.
- Hide underlying complex data structures.

Historical Summaries

For historical data that is to be reported on, the historical data should be pre-processed and summary information retained in the database for reporting, where possible.

Pre-Processing Data

The most efficient way to run some reports will be by pre-processing the data before the Jade Report Writer is run. In this technique, the data to be reported on is collated into a transient object (or set of transient objects, as required) and the Jade Report Writer is run over the collated data. In this case, the Jade Report Writer is being used as a formatting tool. You could use dedicated background processes to pre-process the data.

If the pre-processed data is not dependent on the report parameters and it is likely to be used in several reports, consider persistent instances of these classes. This technique is especially useful for summary reports when large amounts of objects need to be sorted into multiple categories, and the totals reported. In this case, the pre-processing method can efficiently sort the selected object using a single retrieval of the object. If the object is sorted into multiple categories using the Jade Report Writer, the object must be retrieved and inspected for inclusion into each of the separate categories. As the amount of data to be inspected increases, the Jade Report Writer overhead increases proportionally.

Pre-processing the data may also be useful where the data needs to be selected based on report parameters and it is not directly selectable using the selection criteria or direct selection will be too costly; for example:

- Class **X1** has a collection of events by date.
- Method to select the last event in parameter range that is of type *X* and return a value based on that event; for example:

```
select(x: String): String;
```

- If multiple events of different types are required (**A**, **B**, and so on), the Jade Report Writer must make multiple calls; for example:

```
select("A");
select("B");
...
```

If the collection of events is large and the **JadeQueryResult** is large, the overhead of running this method in the query will be quite large, because the method is called for each type and the collection must be searched in each call.

By pre-processing the data, the required outputs for a specified instance of class **X1** can be computed in one search of the collection and the results saved for the report output.

Fixing Unexpected Output

Sometimes the Jade Report Writer output is not what you expect to see. Some of the common problems, with hints on what to look at to resolve the problem, are as follows.

- No report output is produced
  - Check the **jadeReportWriter<*x*>.log** and **jrw_dump_<*x*>** files in the **LogDirectory** location for exceptions.
  - A Jade initialization file parameter exists to specify if exceptions are passed back to the caller.
  - If the **jadeReportWriterCheck** method is implemented for classes being reported on, check the execution statistics for **NumDbReads**, **NumAuthorized**, and **NumSecurityCalls**.
  - The current user may not have access to the data being reported on.

  See also **Missing or extraneous detail lines**, in the following list item.
- Missing or extraneous detail lines
  - Turn on the query trace output from the AQE.
  - Check that the query itself contains the objects and paths, as expected.
  - Check that the where clause in the query is as expected.
  - Check that the dictionaries used in the Query Execution phase are as expected.
  - Check the **Inclusive** check box on the **Types** sheet of the Report Query Options form in the Jade Report Writer Designer application, accessed by selecting the **Query Options** command from the Report menu.

    Setting the **Inclusive** option means that data from the type is reported, whether data in associated collections is present or not.

    Clearing the option means that data from the type is reported only when the collection data is also present.
  - If the root collection uses instances, check that there are no extraneous instances that are being reported on.
  - If the **jadeReportWriterCheck** method is implemented for classes being reported on, check the execution statistics for **NumDbReads**, **NumAuthorized**, and **NumSecurityCalls**.

    Some expected results may not have been authorized correctly for the current user.
- Data items are blank
  - This is usually a result of a 1090 *(null access)* or 1086 *(invalid typecast)* exception during the data access.

    For the object for which the data is being reported, check the data access path as defined in the query trace output.

Summary of Performance Tips

The following recommendations may help improve the performance of the Jade Report Writer.

- Summarize the data that is to be reported
- Pre-process data that is to be reported

- Use collections matching the selection keys to optimize retrieval
- Use profile selection to limit selections as early as possible
- Use concurrency strategy **None** with automatic cache coherency, unless data is changing frequently
- Minimize the use of scripts and summaries