pitney bowes

Spectrum Miner™

Version 8.0

Using the Data Build Manager

Copyright

USPS® Notices

Pitney Bowes Inc. holds a non-exclusive license to publish and sell ZIP + 4® databases on optical and magnetic media. The following trademarks are owned by the United States Postal Service: CASS, CASS Certified, DPV, eLOT, FASTforward, First-Class Mail, Intelligent Mail, LACS^{Link}, NCOA^{Link}, PAVE, PLANET Code, Postal Service, POSTNET, Post Office, RDI, Suite^{Link} , United States Postal Service, Standard Mail, United States Post Office, USPS, ZIP Code, and ZIP + 4. This list is not exhaustive of the trademarks belonging to the Postal Service.

Pitney Bowes Inc. is a non-exclusive licensee of USPS® for NCOA^{Link}® processing.

Prices for Pitney Bowes Software's products, options, and services are not established, controlled, or approved by USPS® or United States Government. When utilizing RDI™ data to determine parcel-shipping costs, the business decision on which parcel delivery company to use is not made by the USPS® or United States Government.

Data Provider and Related Notices

Data Products contained on this media and used within Pitney Bowes Software applications are protected by various trademarks and by one or more of the following copyrights:

ICU Notices

Table of Contents

C o p y r i g h t

1 - Introduction

About this document 6
Spectrum Miner Overview 6
What is the Data Build Manager? 6
Why use the Data Build Manager? 7

2 - Running a data build

Introduction 11
Using the graphical user interface 11
Using the command-line interface 12
Troubleshooting 13

3 - Writing a build plan

Introduction 17
Getting started 17
Build plan structure 22
Choosing an editor 26
Tips and traps 28

4 - Spectrum Miner data-build tasks

Introduction 33
Common attributes and shortcuts 33
Temporary foci 35
Nested elements 36
Importing data 38
Exporting data 43
Combining foci 48
Enhancing foci 52
Transforming foci 55
Managing foci 63
Reporting tasks 69
Building models 76
Other tasks 77

5 - Standard and third-party tasks

Standard Ant tasks 87
Third-party tasks 90

Appendix 92

Appendix A: Appendix

Credits 94

1 - Introduction

In this section

About this document 6
Spectrum Miner Overview 6
What is the Data Build Manager? 6
Why use the Data Build Manager? 7

About this document

This book provides a tutorial and reference guide for the *Data Build Manager* component of Spectrum Miner.

Spectrum Miner Overview

Spectrum Miner is a powerful predictive analytics solution that enables customer insight professionals and business users alike to achieve a clear picture of their customers for the purpose of greater customer understanding, uncovering areas of opportunity, achieving optimal segmentation and predicting future behavior.

Bridging the gap between standard Business Intelligence tools with a limited scope for exploring data, and number-crunching solutions which require statistical programmers to build queries and produce models, Spectrum Customer Analytics is a next-generation solution designed for unparalleled ease of use – and fast actionable insight.

The solution utilizes powerful 3D data visualization and rapid modeling automation to uncover important data relationships and deliver propensity scores at the push of a button, boosting predictive model accuracy and increasing the speed of analytic results.

Spectrum Miner can be used to predict profit-impacting behaviors and propensities, including customer churn, cross sell and up sell opportunities, campaign planning and segmentation, customer satisfaction and loyalty, and customer lifetime value.

What is the Data Build Manager?

The Data Build Manager is a flexible, robust, standards-based data-build utility that supports the real-world data-preparation processes feeding customer analysis, exploration and modeling activities within Spectrum Miner, and subsequent automated deployment of the lists, scores and models resulting from such analysis.

Spectrum Miner provides a suite of *data-build commands* that perform standard operations on flat files, database tables and Spectrum Miner datasets (foci). (For more details, see the **Spectrum Miner Data-build Command and TML Reference Guide**. These commands can be executed manually at the command line, or interactively within Spectrum Miner. However, a typical data build might require a lengthy sequence of operations to generate the final dataset used by an analyst—for

example, data might be sourced from a variety of databases and flat files, then aggregated and joined appropriately to customer level, before overlaying relevant metadata. Performing these steps manually is time-consuming and error-prone. Additionally, since builds often run on a regular basis (for example, each month-end) and can have lengthy runtimes, it's often desirable to skip previously completed steps, incrementally update based on previous runs, and parameterize a build based on user-supplied values.

The Data Build Manager allows the data architect to write a platform-independent *build plan* which describes all of the steps involved in such a data build, including dependencies, conditional logic, parameterization and the like. The build plan itself is an XML document [see **Getting started** on page 17] that describes a build as a collection of *targets*. Each target describes how to make some output product or intermediate component of the build. A target might contain or depend on other targets, or it might contain a sequence of *tasks* that need to be carried out to create the required result. This basic framework makes it easy to implement simple data builds with a no-frills approach while maintaining the power and flexibility to design much more sophisticated customer-oriented ETL (extract-transform-load) processes if required.

The Data Build Manager supports a wide variety of tasks that are useful in data builds—including not just Spectrum Miner data-build commands commands but many standard operations such as: file, directory and archive management; file filtering and preprocessing; executing external processes; concurrent execution of independent operations; e-mail notification; flexible logging; parameterized values including dates and pathnames; remote file and database access; conditional logic and much more.

The Data Build Manager employs Spectrum Miner's Transaction Measurement Language (TML) for concise, powerful aggregation and derivation of new fields.

Once a build plan has been written, it can be executed from the command line or as a scheduled batch job, or interactively via Spectrum Miner. A simple graphical user interface allows detailed control over how the build is executed, including which targets should be updated or skipped, and what performance tuning and logging options to use.

Why use the Data Build Manager?

Customer analysis, data-mining, visualization and exploration typically requires *denormalized* data, that is, one record for each customer (or household or account) with many fields of information summarizing key characteristics of that customer.

Typically the source data is stored in one or more relational databases or flat-file feeds in "normalized" form. That is, many different tables at different levels of detail along with various reference tables containing metadata and lookup information. For example, customer demographics might be stored with one row per customer, while account history might be stored with one row per account per month, and transaction data might be stored with one row per transaction, and so on.

To create a dataset for analysis, a data-build process combines information from the relational tables into a customer view, which involves joining data from related tables and summarizing (aggregating) data from lower levels of detail to the customer-oriented form, as well as incorporating appropriate metadata.

Designing, implementing or modifying an existing data build can range significantly in complexity, depending on factors such as those listed below:

- Analytical requirements:
  - Is there a specific analytical task in mind or is the target dataset intended to support a broad range of analyses?
  - What fields, aggregations and metadata are required for descriptive or predictive analysis?
  - What observation and outcome time periods are required?
- Volume and complexity of source data feeds:
  - How many tables in how many databases are involved?
  - How many fields and records are in the main tables?
  - What are the primary/foreign key relationships between tables?
  - How many levels of "rollup" are required?
- Degree of automation:
  - Should the build run automatically on some regular basis?
  - How much user intervention is required?
  - What parameters will the user specify?
  - What logging and notification is required as the build executes?
  - How should results from previous runs be managed (archival and backup policy)?
- Required performance:
  - Is concurrent execution required to reduce elapsed time?
  - Should the build recover "where it left off" from a previously interrupted or failed execution?
  - Should results from builds in prior time periods feed into the build to reduce runtime (incremental update)?

The Data Build Manager is intended to provide a flexible environment for implementing data builds, regardless of the specific design choices above. It aims to keep simple data builds simple while still supporting even the most complex requirements. The Data Build Manager is designed to be:

- **Simple**: build plans are easy to create and maintain.
- **Robust**: a build plan is executed in a stable, predictable and repeatable way.
- **Complete**: a build is defined entirely by a single build plan.
- **Modular**: a build can be made up from several components, supporting partial builds and handling dependencies between components.
- **Portable**: build plans are platform-independent with non-proprietary syntax.

Key features include:

- **Parameters**: builds can be configured via arbitrary user-supplied parameters, including conditional execution and parameterized outputs.
- **Concurrency**: multiple tasks or targets can be executed in parallel (in addition to the inherent parallelism supported by many of the Spectrum Miner data-build commands).
- **Logging**: all output from tasks is captured and logged, at user-specified level of detail, with e-mail notification supported.

The Data Build Manager is derived from the widely used, open-source **Apache Ant** Java build tool. This brings the advantages of a robust, well-known, highly flexible and open architecture. Although the process of building a customer analysis dataset shares similarities to building a complex software application, the Data Build Manager also provides numerous capabilities specifically targeted at the requirements of customer-oriented data preparation.

Based on more than ten years of experience in implementing and maintaining customer-oriented data builds, Spectrum Software believes the Data Build Manager approach is typically superior to any of the potential alternatives, including generic scripting languages, direct SQL implementations, or dedicated ETL tools.

Various scripting languages can be used for data preparation (awk, Perl, Python, COBOL, . . .) but these are general-purpose programming languages. So although in some sense anything is possible, they lack core concepts about manipulating, linking and summarizing tables of data (in databases or flat files) and thus result in solutions that are lengthy, difficult to support and lack performance without extensive development effort.

SQL and SAS have regularized data manipulation at their core. However, both have several disadvantages. First, they are necessarily somewhat limited in scope (requiring all data to reside in a single database or set of SAS datasets respectively), while at the same time relying on a relatively generic, low-level programming model. This again means that although they have increased flexibility, the resulting solutions are often cumbersome, in terms of complexity, lines of code, and performance, for typical customer-focused data builds.

The Data Build Manager strikes a balance between generality (and its inherent complexity) and simplicity (and its potential limitations) by combining simple, efficient utilities for core data-manipulation operations with a straightforward, XML-based framework supporting the subsidiary tasks often required in the data-preparation process, including flexible parameterization and the option to embed (arbitrarily complex) external scripts or commands within the process.

2 - Running a data build

In this section

Introduction 11
Using the graphical user interface 11
Using the command-line interface 12
Troubleshooting 13

Introduction

This section is aimed at the user of an existing data build. It assumes both a working installation of the Data Build Manager as well as an existing build plan on the Spectrum Miner server platform (typically with a `.qsb` extension, often called simply `qsbuild.qsb`). Builds can be run interactively through Spectrum Miner, using a simple graphical user interface, or from a command prompt on the Spectrum Miner server. Both of these approaches are discussed below. Builds that should execute on a regular basis can be scheduled to run automatically on the Spectrum Miner server, using third-party scheduling tools (contact your Spectrum Miner administrator for details).

Using the graphical user interface

To run a data build using the graphical interface, first open the Spectrum Miner client and log in to the appropriate Spectrum Miner server. When Spectrum Miner appears, browse to the required build plan, and either double-click, or right-click to select the default **Build** option[1].

The interface allows you to browse the build plan's structure[2] and view or modify details of the build. For example:

- Browsing to the top level of the tree shows details of the build itself, including the default target that will be built if you simply select **Build** immediately.
- To change the parameters for the build, browse to parameter elements typically found just under the top-level build element. Use the parameter details panel to view or change the values.
- Browsing to target elements within the build will show the specifics of each target. Right-click the target or use the combo-box in the target details panel to specify whether the target should be built "only if needed" (the default), "always" (forcing a rebuild even if already built), or "skip" (even if another target depends on it).

  Note that when manually selecting multiple independent targets to run, there is currently no way to control the order in which they will execute (of course their individual dependencies will still run correctly). If required, add an appropriate dependency to the build plan or use the command-line interface [see **Using the command-line interface** on page 12] (where targets are executed in the order listed).

[1] If these options are not available or the .qsb file type appears as an unrecognized file type, your Spectrum Miner client might not be configured correctly — contact your Spectrum Miner administrator

[2] Alternatively, since the build plan is simply an XML document, you can browse the build plan structure using most modern web browsers, for example by double-clicking in Spectrum Miner.

- Other elements in the build tree represent component tasks and other modifiers: browsing to these will display the underlying XML to help understand the details of the build structure.

The **Options** menu provides various choices defining how the build will execute—these map directly onto the command-line options, full details of which are in **Spectrum Miner Data-build Command and TML Reference Guide**.

Finally, use the **Build** button to run the build with the parameters and options you've specified, or use **File>Exit** to quit without starting a build.

When a build is executing, a new window will open (below) showing the build progress, which allows you to:

- View build status including start and stop time, and the equivalent command line used to execute it.
- Diagnose any problems: see Section **Troubleshooting** on page 13 for more details. However, note that the `Note: java compilation tasks unavailable...` warning is harmless.
- Cut and paste output from the build. Note that each run also generates a log file named `<smhome>/shared/logs/qsbuild-username-jobnumber.log` (where `<smhome>` is the Spectrum Miner installation directory), which you can access via the **Go To Logs** toolbar button in Spectrum Miner.
- Stop a running build including all active subprocesses using the **Stop** button.
- Detach from a running build (but leave it running on the server) by closing the output window at top right.
- Run multiple builds in different windows (perhaps with different parameters or for different targets). However it is up to the user to ensure that the multiple builds don't conflict with each other.

Using the command-line interface

To run a data build from a command prompt, first open a command window on the Spectrum Miner server (perhaps using remote desktop, **telnet** or **ssh**—contact your Spectrum Miner administrator). Ensure that the environment is configured for Spectrum Miner commands[3].

When your installation is configured correctly, the command `qsbuild -help` should display usage information.

To run a data build, simply type **qsbuild** followed by appropriate command-line options. For example, simply executing **qsbuild** with no options will attempt to build the default target from a build plan called `qsbuild.qsb` found in the current directory, while the command:

[3] This may have been done automatically, or you may need to execute a configuration script such as `<smhome>/config/setpath-qsversion.[bat|sh]` (where `<smhome>` is the Spectrum Miner installation directory).

```
qsbuild -input myproject.qsb -memory 512 -Dmonth=April -skip TooMuchWork
-targets FullBuild
```

will build the target called FullBuild from the build plan found in the XML file `myproject.qsb`, skipping the target `TooMuchWork` even if another target depends on it, with the parameter month set to the value April, and using 512Mb as the default amount of memory for Spectrum Miner data-build tasks.

For full details of the supported command-line options, see **Spectrum Miner Data-build Command and TML Reference Guide**.

Troubleshooting

Capturing output

The first step in diagnosing a problem is to collect detailed information on what happened (or didn't). There are several ways to collect information from the Data Build Manager:

- A standard output log is written either within the user interface [see **Using the graphical user interface** on page 11] output window, or to standard output at the command line. Additional detail can be captured using the `-verbose` option at the command line or within the user interface.
- It is often useful to capture the log output to a file for future reference. The user interface automatically creates a log named `<smhome>/shared/logs/qsbuild-username-jobnumber.log` (where `<smhome>` is the Spectrum Miner installation directory). To capture output at the command line, use the `-logfile` option:

  ```
  qsbuild -logfile somefile.log...
  ```

- Use of diagnostic tasks within the build plan can be invaluable, particularly in tracking down properties that are set (or not) to unexpected values, and to understand when sections of the plan start and finish. For example, see **echoproperties**, **echo**, and **stopwatch**.

Data-build command errors

Errors generated by Spectrum Miner data-build commands typically start with the prefix `*** Error:` or `*** Warning:`. There are several benign warnings that can safely be ignored, including:

- qsimportdb [see **qsimportdb** on page 38] ODBC warnings, such as:

```
  [qsimportdb] *** Warning: SQLConnect(): [unixODBC][FreeTDS][SQL
Server]Changed lan- guage setting to us_english.
  [qsimportdb] *** Warning: SQLSetStmtAttr(): [FreeTDS][SQL Server]Option
 value changed
```

- FDL warnings such as missing keys from `dblookup()` or division-by-zero generating NULL results

Other warnings, and most `*** Error:` reports likely indicate a problem, including examples like these:

```
  [qssort] *** Error: Insufficient memory to sort data.
  [qsimportfocus] *** Error: Failed to execute inline TML
 [qs<anydbc>] *** Error: Cannot read from focus file <somefocuspath>.ftr
```

Build failure messages

The source of failure during a build is reported by lines in the log file like those below. Note however that the root cause of the failure may have occurred earlier depending on prevailing `failonerror`, `-warn` and parallelism settings.

```
BUILD FAILED
D:\qsdata\qsbuild__ant__.xml:1426: Target score failed: qsimportdb
returned: 1
```

The failing task (qsimportdb [see **qsimportdb** on page 38] in this example) is identified along with its exit status (the value 1 here). The failing line in the build plan is reported as line 1426 of `qsbuild__ant__.xml`, the preprocessed version of the original build plan, `qsbuild.qsb`. To find the corresponding location in the original build plan, open the preprocessed file and go to the line indicated. Search backward until you find a comment like this:

```
<!--Begin ~create_report at Line 1309 of ...qsbuild.qsb-->
```

This shows the location of the matching target in the original build plan. If you search forward from that point in the original plan, you will find the code corresponding to the failing line(s) in the preprocessed version.

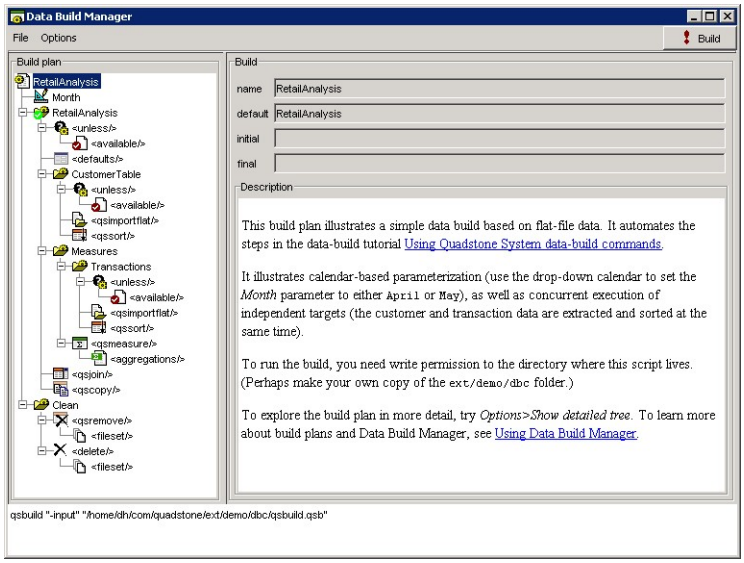

Figure R–1: Data Build Manager user interface

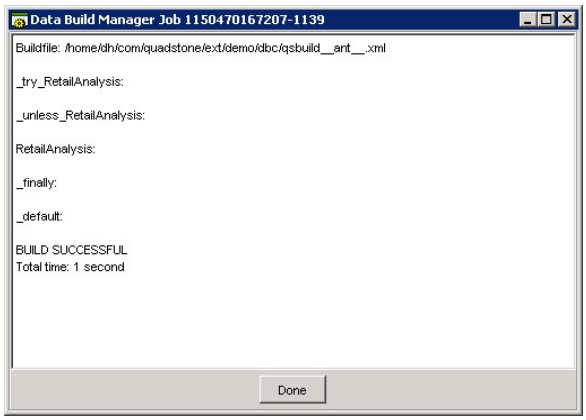

Figure R–2: Data Build Manager output window

3 - Writing a build plan

In this section

Introduction	17
Getting started	17
Build plan structure	22
Choosing an editor	26
Tips and traps	28

Introduction

This section describes how to create a build plan. It assumes basic familiarity with XML [see **Getting started** on page 17], as well as a working Data Build Manager installation on the Spectrum Miner server (see the README file included with the Data Build Manager distribution or contact your Spectrum Miner administrator).

Getting started

A build plan is an XML document on the Spectrum Miner server that defines the steps necessary to create the required output(s) of a data build. By convention, a build plan has a `.qsb` extension. By default the **qsbuild** utility looks for a build plan called `qsbuild.qsb` in the current directory. To create a build plan, simply create an XML file on the Spectrum Miner server with the desired name in the desired location (see Section **Choosing an editor** on page 26 below).

XML is a ubiquitous syntax for storing structured data. If you've never heard of XML, a good place to start is **http://www.w3.org/XML/1999/XML-in-10-points**. Other useful links can be found at **http://www.xml.org/xml/resources_focus_beginnerguide.shtml**. An XML document is made up of elements, enclosed in angle brackets (< >), which contain text or other elements, and can include attributes which are simple name/value pairs. For example:

```
<book>
  <author name="Joe Bloggs"/>
  <chapter title="Chapter 1">
      It's a short story.
  </chapter>
</book>
```

is a simple document with a `book` element containing two child elements: an `author` element with a **name** attribute and no content, and a `chapter` element with a **title** attribute and some text content. (Also see Section **Special characters in XML** on page 25.)

A build plan is contained within a single build [see **The build element** on page 22] element, which itself contains at least one target [see **The target element** on page 23] element. Each `target` element contains a sequence of nested elements that specify the sub-targets or tasks required to create some result of interest. Each task is represented by a specialized XML element that defines the characteristics of that task using appropriate attributes and nested elements. For example, this

simple build plan, `qsbuild-starter.qsb`[4], uses the qsimportflat [see **qsimportflat** on page 39] task to build a focus from a flat file:

```
<?xml version="1.0" ?>
<build name="RetailAnalysis" default="CustomerTable">
  <description>
  This is an example build plan.
  </description>

  <target name="CustomerTable" description="Import the customer data">
    <qsimportflat input="RetailCustApril.txt"
output="RetailCustApril.ftr" logfile="RetailCustApril.log"/>
  </target>

</build>
```

(The first line in the document is optional[5], indicating to other applications that the document is XML, and will be omitted from subsequent examples.) This plan could be executed either by double-clicking on it in Spectrum Miner or typing **qsbuild -input qsbuild-starter.qsb** at a command prompt on the server.

We will explore the capabilities of the Data Build Manager by creating a build plan to build an analytical dataset from the demonstration data and example build plans found in `<smhome>/ext/demo/dbc` (where `<smhome>` is the Spectrum Miner installation directory). The source data consists of:

`RetailCustmonth.txt` a monthly flat-file extract of customer data, where month is `April` or `May`. The file contains one row per customer with a key field called `CustomerID`.

`RetailTransmonth.txt` a monthly flat-file extract of sales transactions for the customers. The file contains zero or more records per customer, also keyed on `CustomerID`.

The goal is to aggregate the transactional data to create customer-level measurements, and join to the customer data to create a monthly analytical view. The simplest build plan structure simply executes the steps sequentially:

```
<build>
  <description>A simple sequential databuild</description>

  <parameter name="Month" value="April" format="%B" type="date"
```

[4] This example and the other build plans discussed here can be found in the `<smhome>/ext/demo/dbc` folder (where `<smhome>` is the Spectrum Miner installation directory on the server). (In Spectrum Miner use the Go to Logs toolbar button and go up two levels to find `<smhome>`.)

[5] However, it is recommended. For example, if Internet Explorer is used to view a build plan, the `<?xml?>` tag enables hierarchical browsing of the document structure.

```
                description="Select month to build"/>

  <target name="RetailAnalysis" description="Build the analysis dataset">

    <defaults keys="CustomerID" memory="128"/>
    <qsimportflat input="RetailCust${Month}.txt" tmp="tmp_cust.ftr"/>
    <qssort input="tmp_cust.ftr" tmp="RetailCust${Month}Sorted.ftr"/>
   <qsimportflat input="RetailTrans${Month}.txt" tmp="tmp_trans.ftr"/>

    <qssort input="tmp_trans.ftr" tmp="RetailTrans${Month}Sorted.ftr"/>

    <qsmeasure input="RetailTrans${Month}Sorted.ftr"
                  tmp="RetailTrans${Month}Measures.ftr"
                  library="functions.tml">
      <aggregations>
        create numberPurchases := ( count() );
        create totalAmount := ( sum(Amount) );
        create mostCommonStore := ( mode(Store) );
        create totalPointsRedeemed := ( sum(PointsRedeemed) );
        create averageSpendInStore_ := ( mean(Amount) )
        by ( StoreSplitFunction( Store ) );
      </aggregations>
    </qsmeasure>
    <qsjoin input="RetailCust${Month}Sorted.ftr"
              join="RetailTrans${Month}Measures.ftr"
              tmp="Retail${Month}Join.ftr"/>
    <qscopy from="Retail${Month}Join.ftr"
to="Retail${Month}Analysis.ftr"/>
  </target>
</build>
```

This build plan illustrates a number of features of the Data Build Manager:

- A property called month is declared as a parameter [see **parameter** on page 79] with a default value of `April`, and then referenced in the names of input and outputs using the `${Month}` syntax. The user of the plan has the opportunity to modify parameters before executing the build, making it easy to build a dataset for `May` using the same build plan. Parameter type and format hints let the user interface provide appropriate selectors and validation for files, directories and dates.
- Several default attribute/value pairs are defined in the defaults [see **defaults** on page 78] element. These are used to infer the values for unspecified attributes in subsequent tasks, for example all of the Spectrum Miner data-build tasks will inherit the **memory** attribute, and qssort [see **qssort** on page 62], qsmeasure [see **qsmeasure** on page 56], qstrack [see **qstrack** on page 58], and qsjoin [see **qsjoin** on page 48] will inherit the **keys** attribute.
- To avoid proliferation of temporary foci, the intermediate tasks in the build use the **tmp** attribute instead of **output** or **to** to specify the output focus. These temporary foci [see **Temporary foci** on page 35] are automatically removed at the end of the top-most enclosing target (unless the build fails, as they are often useful to diagnose what went wrong).

- The qsmeasure [see **qsmeasure** on page 56] and qstrack [see **qstrack** on page 58] tasks in this case specify a function library using an attribute that points to a TML file, but embed the TML for the measurements inline using a nested aggregations [see **Inline scripting** on page 36] or trackers [see **Inline scripting** on page 36] element.

Although probably unnecessary in such a simple example, we might choose to add additional features to the build, resulting in this plan:

```
<build name="RetailAnalysisDB" default="RetailAnalysis">
  <description>
<p>
This build plan illustrates a simple data build based on
tables stored in an Access data source.
</p>
  </description>

  <parameter name="Month" value="April" format="%B" type="date"
             description="Select month to build"/>

  <target name="RetailAnalysis" description="Build the analysis dataset">

    <unless><available file="Retail${Month}Analysis.ftr"/></unless>
    <defaults keys="CustomerID"/>
    <target name="CustomerTable" concurrent="true">
     <unless><available file="RetailCust${Month}Sorted.ftr"/></unless>

      <qsimportdb udc="user@retail" table="RetailCust${Month}"
                  tmp="tmp_cust.ftr"/>
      <qssort in="tmp_cust.ftr" tmp="RetailCust${Month}Sorted.ftr"/>
    </target>

    <target name="Measures" concurrent="true">
      <target name="Transactions">
     <unless><available file="RetailTrans${Month}Sorted.ftr"/></unless>

      <qsimportdb udc="user@retail" table="RetailTrans${Month}"
                  tmp="tmp_trans.ftr"/>
      <qssort in="tmp_trans.ftr" tmp="RetailTrans${Month}Sorted.ftr"/>
    </target>
    <qsmeasure in="RetailTrans${Month}Sorted.ftr"
               tmp="RetailTrans${Month}Measures.ftr"
               library="functions.tml">
      <aggregations>
        create numberPurchases := ( count() );
        create totalAmount := ( sum(Amount) );
        create mostCommonStore := ( mode(Store) );
        create totalPointsRedeemed := ( sum(PointsRedeemed) );
        create averageSpendInStore_ := ( mean(Amount) )
        by ( StoreSplitFunction( Store ) );
      </aggregations>
    </qsmeasure>
```

```
  </target>
  <qsjoin input="RetailCust${Month}Sorted.ftr"
          join="RetailTrans${Month}Measures.ftr"
          tmp="Retail${Month}Join.ftr"/>
  <qscopy from="Retail${Month}Join.ftr" to="Retail${Month}Analysis.ftr"/>

  </target>

  <target name="Clean" description="Remove all foci and temporary files">

    <qsremove><fileset dir="." includes="*.ftr"/></qsremove>
    <delete>
     <fileset dir="." includes="*~,*.fdd*,*.log" defaultexcludes="no"/>

    </delete>
  </target>

</build>
```

The key changes in the build plan are:

- The source data is being imported from an ODBC source called `retail`, exposed in Spectrum Miner as a user database connection (UDC) called `user@retail` configured with the **qsdbaccess** command (see the **Spectrum Miner Data-build Command and TML Reference Guide**).
- The build [see **The build element** on page 22] element now contains more than one target, so now must specify the default target (`RetailAnalysis`). The other target (`Clean`) is an administrative target that a user can execute to force a build from scratch.
- The `RetailAnalysis` target contains an initial unless [see **The target element** on page 23] condition. If the condition is met then the target will not be executed. Typically targets contain checks that short-circuit them if their final output is already up-to-date.
- The `RetailAnalysis` target now contains nested targets (`CustomerTable` and `Measures`, which itself contains `Transactions`), rather than a linear sequence of tasks. These help to structure the build plan into functional units.
- The two child targets of `RetailAnalysis` specify **concurrent=**"`true`", specifying that that they should execute concurrently [see **Concurrency** on page 24] with one another (the subsequent qsjoin [see **qsjoin** on page 48] task will not start until both are complete).
- The additional tasks qsremove [see **qsremove** on page 67] and delete have been introduced.

Build plan structure

This section describes the structure of a build plan in more detail. It draws heavily on the **Apache Ant documentation**[6]. The Data Build Manager distribution also includes a schema that precisely defines the structure of a legal build plan. Although the schema is primarily intended to support plan validation and schema-aware XML editors, it is also sometimes useful to resolve subtle syntax issues. Note that in contrast to **Ant**, the schema is case-sensitive: all task and attributes names must appear entirely in lowercase.

The normative schema is defined in **RELAX NG compact syntax**. A **W3C XML Schema** version is also available but is in some areas less strict.

XML files created by Spectrum Miner automatically include an optional schemaLocation attribute that points to schemas that you can access directly from your Spectrum Miner installation at:

`<smhome>/server/qs8.0/integration/schemas` (where `<smhome>` is the Spectrum Miner installation directory)

You can also find examples of how to process these XML files at:

`<smhome>/server/qs8.0/etc/xslt`

The build element

A build plan is an XML document with a single `build` element containing:

- an optional `description` element containing a human-friendly description of the build, for display by the user interface [see **Using the graphical user interface** on page 11] or the `-describe` command-line option. The description can contain text and/or HTML markup[7] including active hyperlinks.
- an optional defaults [see **defaults** on page 78] element specifying default values for common attributes in the remainder of the build plan.
- (optionally) one or more Ant **property** elements specifying internal build properties
- (optionally) one or more parameter [see **parameter** on page 79] elements defining user-configurable build properties (note these never occur inside targets, unlike `property` elements)
- one or more target [see **The target element** on page 23] elements that define components of the overall build.

[6] Several books with perhaps more approachable introductions to Ant are available, including the *Ant Developer's Hand-book* (ISBN 0-672-32426-1).

[7] Note that `CDATA` is not valid HTML: use `PRE` or entity references to embed special characters [see **Special characters in XML** on page 25].

The `build` element is identical to Ant's **project** element, with these changes:

- An optional **initial** attribute names a target that should always be executed before any other target(s), that is, an implicit dependency of every target. (But if your initial target calls or depends on any other target that isn't nested within it, that will trigger your initial target again and your plan will enter an endless loop—so make sure you nest any dependencies within the initial target itself.)
- An optional final attribute names a target that should always be executed after the build completes or fails. If the build fails, a property called `_fail_message` will contain the failure message (it will otherwise be unset).
- If a `build` element contains only a single target (which itself may contain nested `targets`), the default attribute can be omitted.

The target element

Each `target` element defines the sequence of tasks required to build that target. These are modeled on Ant's **target** element with these differences:

- A `target` lement may contain an optional boolean concurrent [see **Concurrency** on page 24] attribute.
- The Ant `target` element's **if** and **unless** attributes are replaced by an optional initial `unless` element. The `unless` element is like Ant's **condition** task but without a **property** attribute. If the condition evaluates to `true`, the target (and all dependencies) are considered up-to-date and thus not executed. The `unless` element can optionally contain a **message** attribute which is output if the condition is true (that is, when the target is skipped). For example:

```
<target name="Output">
  <unless message="Output already built"><available
file="Output.ftr"/></unless>
  ...
```

  will skip the `Output` target and print a message if the focus `Output.ftr` already exists.

- The remainder of the target consists of an intermixed sequence of nested `target` elements and dependency [see **The dependency element** on page 23] elements (which replace the depend attribute on Ant's `target` element).

The dependency element

The `dependency` element indicates that a target depends on some other target, using the syntax `<dependency target="SomeTarget"/>`. It may optionally contain a concurrent [see **Concurrency** on page 24] attribute. Dependent targets are only executed once even if multiple

targets depend on them (in contrast to targets that are explicitly invoked using qscall [see **qscall** on page 79], possibly with different parameters).

Concurrency

An important feature of the Data Build Manager is its ability to execute multiple tasks at the same time (concurrently). This can improve performance by utilizing more server resources (of course the reverse can occur if too many jobs cause contention for resources!).

Both target [see **The target element** on page 23] and dependency [see **The dependency element** on page 23] as well as all task elements support the boolean concurrent attribute (defaulting to `false` if not specified). This has the following effects:

- Each consecutive sequence of `target` and `dependency` elements, *or* sequence of task elements with `concurrent="true"` will be processed concurrently. (A mixed sequence of tasks and targets are not processed concurrently, even if all are marked concurrent.)
- Makeing a `target` or `dependency` as concurrent does *not* imply that its children will execute concurrently, simply that it will be executed concurrently with other targets or dependencies at the same level.

The build [see **The build element** on page 22], target [see **The target element** on page 23] and **foreach** tasks support a **maxthreads** attribute that controls the maximum degree of concurrency for its nested elements. For example, a target with `maxthreads="2"` would execute at most two jobs at a time in any concurrent sequence of tasks or child targets. The `build` and `target` elements (but not `foreach`) also support a **maxthreadspercpu** attribute which controls the maximum degree of concurrency as a function of the number of physical processors on the server (for example, a target with `maxthreadspercpu="2"` would execute four jobs at a time on a two-processor server).

Other basic concepts

A wide variety of task elements are provided for use within the targets and (sometimes) within other tasks. A task typically performs some external command, and/or manipulates properties. **Properties** provide an environment for the build, and are typically embedded in attribute values, elements or files using `${propertyname}` syntax. Note that, as in Ant, properties are immutable—that is, once set, they can never be changed (except that they can be overridden by parameters when a target is invoked through qscall [see **qscall** on page 79]).

Data Build Manager tasks have the same **common attributes** as Ant tasks (id, taskname, description), along with the optional **concurrent** attribute described above.

Ant defines various **data-type** elements that can be nested within some task elements, for example representing pathnames, wildcards, etc. These carry over unchanged for appropriate tasks, and are supported by Spectrum Miner data-build tasks where appropriate. A qsdateproperty [see

qsdateproperty on page 80] task is provided to manipulate date values stored as strings, but there is no integral date type.

Special characters in XML

Occasionally element and attribute content will need to contain characters such as <, >, ", and & which are part of the XML markup syntax. To insert such characters, they can be replaced by XML entity references using the syntax below (for example, < inserts a < symbol). Alternatively, element content text can be enclosed in a CDATA section which ignores special characters by preceding it with `<![CDATA[` and following it with `]]>`. Also note that attribute values, though normally shown within double quotes, can be enclosed by single quotes in order to embed a double-quote character.

Entity reference	Literal value
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

For example, `<qsimportflat records="Age &gt; 18" .../>` imports records where Age > 18; `<qsimportflat records='MaritalStatus = "single"' .../>` embeds a double-quoted string in a single-quoted attribute, and the following fragment shows the use of CDATA to embed literal text which ignores special characters like the less-than symbol:

```
<qsderive input="cust.ftr" output="cust_deriv.ftr">
  <derivations>
<![CDATA[
    create Rich := Income > 100000;
]]>
  </derivations>
</qsderive>
```

Choosing an editor

Because a Data Build Manager build plan is an XML document [see **Getting started** on page 17], there are a wide variety of choices for creating and editing build plans, from simple text editors through schema-aware XML editors. Any basic text editor will work in a pinch (**notepad**, **vi**, ...), but numerous XML editors that "understand" XML syntax greatly simplify the task of writing a build plan without having to worry about matching angle-brackets and so forth. Some XML editors are also "schema-aware" in that they not only understand standard XML syntax but also read and respect the rules that define which elements and attributes are valid within a specific document type. The Data Build Manager provides both RELAX NG and W3C schemas [see **Build plan structure** on page 22] describing a valid build plan.

Two good, freely available choices are:

jEdit combines text editing with a linked XML structure browser for easy navigation (see Figure **R–1**). Features include: folding to hide/show details of XML elements; remote editing of server files from a client PC via FTP or SFTP; schema-aware validation and wizard showing legal attributes/elements; highly customizable with many powerful editing features and a wide variety of plugins. See Section **Working with jEdit** on page 26 for step-by-step setup instructions.

Emacs is a uniquely customizable text editor, including various modes that support editing of XML content. One of the best is **nXML mode**, which is schema-aware and non-intrusive (see Figure **R–2**). It validates your build plan on the fly and highlights both XML syntax errors and schema correctness. Recommended for users with previous Emacs experience; new users may find it a fairly steep learning curve. Contact **software.support@pb.com** for setup instructions.

Commercial choices include:

- the **oXygen XML editor**
- **XMLSpy**

Working with jEdit

Installing and configuring jEdit

1. Download and install jEdit from **http://www.jedit.org/**
2. Open jEdit and go to the **Plugins>Plugin Manager>Install** tab (it will take a few seconds to populate)
3. Check the XML, XMLIndenter, FTP and Jdiff plugins (they are organized by type rather than alphabetically) and click **Install**.
4. Click Plugin Options and set these options:

- **SideKick>General**: check Parse on keystroke
- **XML>General**: change Show tag attributes in tree to All.

5. Close the plugin manager.
6. Open **Utilities>Global Options>Editing**.
   - set **Folding** mode to sidekick
   - set **Word** wrap to soft
7. Open **Plugins>SideKick>Structure Browser**. Click on the pull-down triangle in the upper-left corner of the window and select **Dock at Right**.
8. If desired, dock **Plugins>ErrorList>Error List** to show XML validation errors.

Using jEdit:

- Use **File >Open** to open your build plan[8]. If the plan is stored on a remote machine (for example, the Spectrum Miner server) use **Plugins>FTP** from the file browser or main menu to connect to and open files from a standard or secure FTP server[9].
- Use the down triangles in the left gutter to fold/unfold sections of the document.
- Use the right-hand navigation window to browse the document structure, or click on an element to move to that element in the document.
- Browse the extensive **Help>jEdit Help** to learn more about jEdit's functionality.

To enable on-the-fly validation of your build plan in jEdit, simply declare the Data Build Manager schema location on the build element like this (adding the `xmlns:xsi` and `xsi:noNamespaceSchemaLocation` attributes):

```
<build
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://www.quadstone.com/xml/qs8.0/qsbuild.xsd"

  name="ExampleBuild"
  default="BuildEverything">

 <target name="BuildEverything">
   ...
```

jEdit will automatically download the schema definition from the URL above and begin validating your document. The **Error List** plugin allows you to find and correct errors, and the **Plugins>XML>**

[8] There is an example build plan in `<smhome>/ext/demo/dbc/qsbuild.qsb` (where `<smhome>` is the Spectrum Miner installation directory)

[9] You may need to ask your system administrator to enable FTP or SFTP on the server.

XML Insert window provides context sensitive assistance for inserting and editing legal elements and attributes at any point in your plan[10].

Tips and traps

This section explores a number of common design patterns and best practices that may be useful in constructing build plans.

- Use the `build's<description/>` element and `target's`**description** attribute to document your build, since this documentation is passed on to the user through the graphical interface and `-describe` command-line option. Of course, XML comments (enclosed by <!-- and --> pairs) are also useful to describe plan complexities for future maintenance, but these are hidden from end users.
- The various mechanisms for calling targets indirectly (for example, dependency [see **The dependency element** on page 23], runtarget, qscall [see **qscall** on page 79] and foreach) differ in the way they pass and set property values between the calling and called targets. The first two execute a target in the existing build context, so that the called target sees (and sets) the same properties as the caller. The latter two use a new build context, so the called target may or may not see the parent properties (depending on the settingof the **inheritall** attribute, and whether they have been overridden by passed parameters) and cannot "return" property values to the caller (since any properties set in the new context are lost once it completes).
- The foreach looping construct is a powerful tool for performing a similar set of tasks on a list of files (or other items). The following example illustrates processing six monthly files and then merging them together: although `foreach` can't directly return a value from each iteration, it can write output to a file which is then read by the caller:

```
<property name="currentMonth" value="20040601"/>
<!-- build a set of monthly files for last six months -->
<foreach list="1,2,3,4,5,6" target="BuildMonth" param="monthOffset/>
<!-- read 'returned' file names into properties month1, month2, ...
month6
<property file="monthlyfiles.properties"/>
<!-- then perhaps merge the files together -->
<qsmerge merge="${month1},${month2},...,${month6}" output="..."/>
 ...

<target BuildMonth>
     <!-- figure out the file name for ${monthOffset} months ago -->
     <qsdateproperty property="fileName" value="${currentMonth}"
             outputpattern="'Month'%Y%m%d'.ftr' "inputpattern="%Y%m%d"
```

[10] Note that the definitive build plan schema is the RELAX NG version; the W3C version is less strict in some places.

```
            offset="-${monthOffset}" unit="month"/>
     <!-- build a focus at ${fileName} -->
     ...
     <!-- return the file name to the caller via a property file
      by associating it with a property 'monthN' -->
     <propertyfile file="monthlyfiles.properties">
        <entry key="month${monthOffset}" value="${fileName}"/>
     </propertyfile>
</target>
```

This same trick can even be used to generate repetitive TML scripts for a large set of fields from a simple template:

```
<!-- make sure output doesn't exist yet -->
<delete quiet="true" file="derivs.tml"/>
<foreach param="field" trim="true" list="ChkBal, SavBal, LoanBal, ...">

   <tasks>
     <!-- append the expanded template to the derivations file for each
 field -
>
     <echo append="true"
       file="derivs.tml">create ${field}_ratio := ${field}_3M /
${field}_6M;</echo>
   </tasks>
</foreach>
<!-- apply the derivations to the input data -->
<qsderive input="monthly_averages.ftr"
   output="monthly_ratios.ftr" derivations="derivs.tml"/>
```

- To increase turnaround during initial development and later enhancements to your build plan, implement your build so that you can easily create and switch between full-size and sampled data. For example:

```
<target name="MakeSamples">
   <foreach param="focus" list="customer,demogs,txns">
       <tasks>
          <unless><available file="samples/${focus}.ftr"/></unless>
          <!-- extract customers with IDs starting with 99 -->
          <qsimportfocus input="extracts/${focus}.ftr"
                         output="samples/${focus}.ftr"
                         records="substr(CustomerId,0,1) = "99"/>
       </tasks>
   </foreach>
</target>
```

Then a simple `${sourcedata}` property can switch the remainder of the build between sample and full-size by assigning `samples` or `extracts` respectively.

- If a plan reuses a number of "subroutine" targets that are never directly executed, they can all be embedded in a dummy `Subroutines` target that always short-circuits to simplify the plan structure. For example:

```
<target name="Subroutines">
   <unless message="These targets are only called indirectly">
      <istrue value="true"/>
   </unless>
   <target name="SomeRoutine">
      ..
   </target>
    ...
</target>
```

- Using property files to set multiple properties via `<property file="..."/>` or **loadproperties** is a powerful way to allow external build parameterization and localization. Because properties are immutable (that is, fixed as soon as they are first set), a plan can load multiple property files which each have the opportunity to override a property defined in a later file (or the build plan itself). A common technique is to load a `local.properties` file before a `standard.properties` file. The first might not even exist by default, but gives the user the ability to override any of the standard properties without modifying any existing files. Uses might include having different setups on a development vs. production platform.

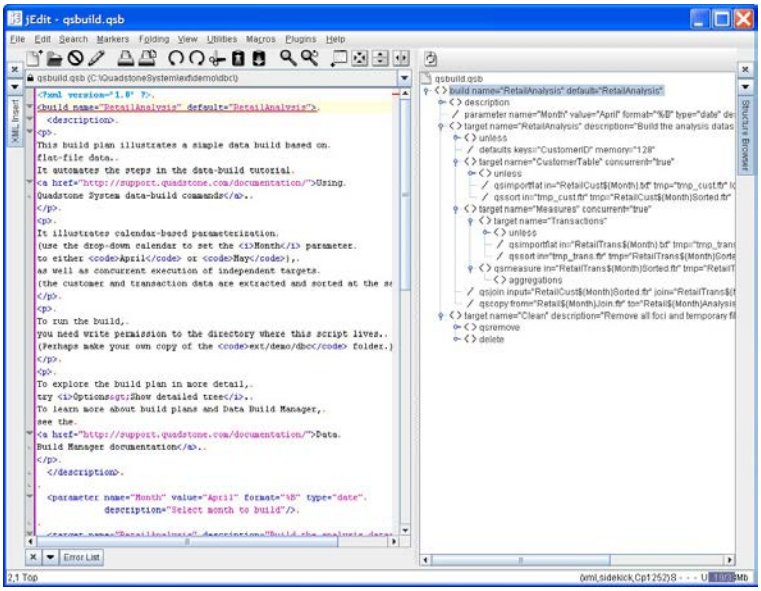

Figure R–1: jEdit user interface with XML support

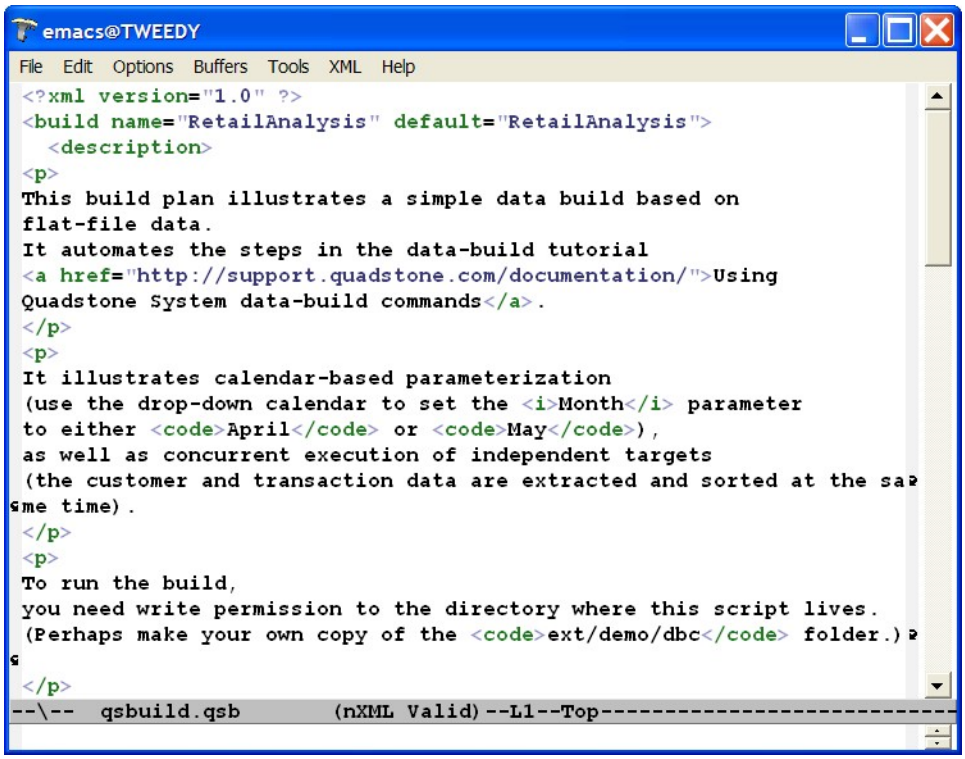

Figure R–2: Emacs editor with nXML support

4 - Spectrum Miner data-build tasks

In this section

Introduction	33
Common attributes and shortcuts	33
Temporary foci	35
Nested elements	36
Importing data	38
Exporting data	43
Combining foci	48
Enhancing foci	52
Transforming foci	55
Managing foci	63
Reporting tasks	69
Building models	76
Other tasks	77

Introduction

The Data Build Manager provides a number of new tasks to support Spectrum Miner data-build commands. They are described in more detail in subsequent sections.

Data Build Manager tasks have the same **common attributes** as Ant tasks (**id**, **taskname**, **description**), along with an optional concurrent [see **Concurrency** on page 24] attribute. Most tasks take attributes which map directly to the command-line options of the corresponding data-build command. One common exception is where a data-build command supports both `-option` value and `-option @filename` with the second reading the option value from a file. This typically maps to **option** and **optionfile** attributes in the task. Also note that every attribute requires a value: whereas a data-build command might use an option without a value to enable some feature (such as `-email`) the corresponding XML form would be a boolean attribute taking the value `true` or `false` (for example, `<qstaskname email="true" .../>`).

The **qsdbaccess**, **qsdescribelicense**, and **qssettings** data-build commands are *not* supported tasks, since they are typically used only interactively. For example, UDC database credentials should be created prior to build-plan execution using **qsdbaccess** at the command line. The other unsupported data-build commands are the QMML-processing utilities **qslt**, **qsqmmledit**, and **qsqmmlview**. (Note however that the **exec** task can be used to execute arbitrary commands.)

Common attributes and shortcuts

With the exception of qsdateproperty [see **qsdateproperty** on page 80], all Spectrum Miner data-build tasks support the standard attributes described in the table below. For convenience the attributes **input** and **output** may be used interchangeably with the synonyms in and out when referring to foci[11].

Attribute	Required?	Notes
Attributes based on standard data-build command options		
config	N	Use this additional configuration file. Alternatively, a configuration file can be specified inline within a nested config [see **Inline scripting** on page 36] element.

[11] The short forms in and out will be deprecated in a future release.

Attribute	Required?	Notes
email	N	Boolean. If "true" send an e-mail message upon completion, using pre-configured e-mail preferences.
logfile	N	Write output to this log file.
memory	N	Use at most the specified amount of memory. The command will try to keep within the limit if possible, but will definitely not exceed it. In cases where a command requires more memory than the maximum amount, the command fails.
parallel	N	Use this many processors to execute the command.
progress	N	Boolean. Show detailed progress reporting
settings	N	Use this settings file
statusfile	N	Write an **HTML** summary to this file.
verbose	N	Boolean. Show progress and other diagnostic information.
Attributes not corresponding to command-line options		
dir	N	The working directory in which the command should be executed. If not specified defaults to the build directory.
failifexecutionfails	N	Boolean. Stop the build if we can't start the program. Defaults to `true`.
failonerror	N	Boolean. Fail if command exits with non-zero status. Defaults to `true`.
outputproperty	N	Store any command output in a property with this name.
resultproperty	N	Store the exit status of the command in a property with this name. (Only relevant if failonerror is `false`.)
timeout	N	Stop the command if it doesn't finish within the specified time (given in milliseconds).

Several of these settings can be defaulted by defining global properties in the build plan, using the reserved property names below:

Property Name	Description
qsbuild.dir	Defines the default directory for all Spectrum Miner data-build commands (*does not affect other Ant tasks*). If not set this will default to the directory where the build plan is located.
qsbuild.failonerror	Determine whether Spectrum Miner data-build command failures should cause the build to stop. If set to false a failing data-build task will not stop build, unless the task explicitly overrides this setting. Defaults to true.
qsbuild.timeout	Define a time in milliseconds after which Spectrum Miner data-build tasks should be killed. Setting this property is not recommended.

Other Spectrum Miner-specific properties

The Data Build Manager sets the values of these other properties:

Property Name	Description
qsbuild.file	The name of the build plan, as it was specified to **qsbuild** on the command line.
qsbuild.historylog	The pathname of the log file `<smhome>/shared/logs/qsbuild-username-jobnumber.log` (where `<smhome>` is the Spectrum Miner installation directory).

Temporary foci

Any data-build task that produces an output focus via the output or to attribute can substitute the tmp attribute in place. This will still generate the specified output focus, but will mark it for removal at the end of the top-most `target` that encloses the task [12].

[12] Note: when the temporary focus is removed, its name will be re-evaluated, that is, re-expanding embedded property values. This could have unintended effects if for example `qscall` is used to call a nested target different values for an embedded property. It's bad practice to call a nested target in any case!

Note: this means that if you use the `-targets` option to execute a nested target, without also executing its top-most enclosing target, its temporary foci will not be removed

If a build creates all intermediate foci using tmp rather than output or to, and the final output(s) don't depend on data in those temporary foci[13], the Data Build Manager will automatically qsremove [see **qsremove** on page 67] all the temporary foci in reverse order of creation when each top-level target completes successfully. If a build fails, all existing temporary foci will be preserved for diagnostic purposes.

If the runtime `-debug` option is specified, all temporary foci will be preserved.

Nested elements

The Spectrum Miner data-build tasks share some new nested elements, described below. Other supported nested elements such as **fileset** and **mapper** are already defined as basic **Ant Concepts and Types**.

macro

This nested element is used by qsderive [see **qsderive** on page 55], qsmeasure [see **qsmeasure** on page 56], qsselect [see **qsselect** on page 61], qstrack [see **qstrack** on page 58], and supports textual substitution in **TML** and **FDL** input files equivalent to Ant's **property** task, using the same `${macroname}` notation. The `macro` element is useful for defining local substitutions or substitutions in text files referenced via attributes. Note that Ant properties are also expanded within nested text elements, but not in files referenced by attribute names (unless explicitly via a **filterchain** containing the `expandproperties` task).

Attribute	Required?	Notes
name	Y	name of macro
value	Y	value for named macro

Inline scripting

Several data-build commands require scripts to define field derivations, aggregations, configuration details, and so forth. While such scripts can typically be referenced in an external file through an

[13] If the final focus does depend on a prior temporary, use qscopy [see **qscopy** on page 63] to break the dependency

appropriate attribute, most such tasks also allow the scripts to be embedded inline within the build plan using an equivalent nested element. The following nested elements all have the same structure: `library`, `aggregations`, `trackers`, `tml`, `sql`, `derivations`, `statistics`, and `config`. Each element supports the attributes listed in the table below.

The element can contain the script, in text form, as the element content. In this case, the element must be specified *without* a namespace. Note that the text can optionally be enclosed in a CDATA section or escaped with XML entity references if the enclosed text contains XML special characters [see **Special characters in XML** on page 25] such as <, >, ", &.

For tasks that correspond to data-build commands that can accept input in XML form (`qsderive`, `qsimportmetadata`, `qsmeasure`, `qsrenamefields`, `qsselect`, and `qstrack`), a build plan can alternatively use inline scripting in *XML* form. In this case, the element must be defined in the `http://www.quadstone.com/xml` namespace.

Attribute	Required?	Notes
deleteonexit	N	Boolean. Defaults to true. The element contents get generated into a temporary script. This indicates that whether this script should be deleted or not.

Examples:

```
<aggregations>
   create numberPurchases := count(); create totalAmount := sum(Amount);
 create mostCommonStore := mode(Store);
   create totalPointsRedeemed := sum(PointsRedeemed);
   create averageSpendInStore_ := mean(Amount)
   by StoreSplitFunction(Store);
</aggregations>

<sql>
   select * from SRC_NOSCORE order by "CUSTOMER_NUMBER";
</sql>

<config>
   [Audits and snapshots]
   htmlimages = none
</config>

<derivations xmlns="http://www.quadstone.com/xml">
   <field name="random" type="integer">
      <fdl>rndBool()</fdl>
   </field>
</derivations>

<library deleteonexit="false">
   <![CDATA[
```

```
   function isRich(income) {
    if (Income > 100000) then 1 else 0;
   }
]]>
</library>
```

Importing data

qsimportdb

Create a focus from a database table or SQL `SELECT` statement.

Attribute	Required?	Notes
output	Y	Create this focus as output. To overwrite an existing focus, use force.
udc	Y	Use these database credentials.
force	N	Boolean. If `true` allow overwrite of existing output focus.
fields	N	Comma-separated list of fields to import.
fieldsfile	N	File listing fields to import.
xfields	N	Import all fields except these.
xfieldsfile	N	Import all fields except those listed in this file.
catalog	N	Locate the database table in this catalog.
schema	N	Locate the database table in this schema
sql	Must have one of these attributes or nested sql [see **Inline scripting**	Read data from this SQL select expression.
sqlfile		Read data from the SQL select expression in the specified file.

Attribute	Required?	Notes
table	on page 36] element.	Read data from the specified table.

NOTES:

- You cannot use the **fields**, **fieldsfile**, **xfields**, **xfieldsfile**, **catalog**, **schema**, or **table** attributes with an **sql** or **sqlfile** attribute, or with a nested `sql` element.
- A nested `sql` element can be used to specify more complex SQL inline, rather than providing it within an attribute or via an external file. Remember to use `CDATA` or XML entities to escape special characters [see **Special characters in XML** on page 25].

Example (note the use of a single-quoted sql attribute to embed a double-quoted field name):

```
<qsimportdb udc="ecstest@eigg" output="SRC_NOSCORE"
           sql='select * from SRC_NOSCORE order by "CUSTOMER_NUMBER"'/>


<qsimportdb udc="ecstest@eigg" output="SRC_NOSCORE">
   <sql>
      select * from SRC_NOSCORE order by "CUSTOMER_NUMBER"
   </sql>
</qsimportdb>
```

qsimportflat

Create a focus from flat-file data.

Attribute	Required?	Notes
input	Y	Read data from this flat file.
output	Y	Create this focus as output (use **force** to overwrite).
force	N	Boolean. If `true`, allow overwrite of existing output focus.
fields	N	Import only this comma-separated list of fields.
fieldsfile	N	Import only the fields listed in this file.

Attribute	Required?	Notes
xfields	N	Import all fields except these.
xfieldsfile	N	Import all fields except those listed in this file.
maxerrors	N	Abort if more than this number of errors is generated about the format of the flat file.
maxwarnings	N	Abort if more than this number of warnings is generated about the format of the flat file.
nowarnings	N	Boolean. If `true`, do not generate any warnings about the format of the flat file.

Examples:

```
<qsimportflat input="cust.fdd" output="cust.ftr" nowarnings="true"/>
```

qsimportstat

Create a focus by importing records from a Microsoft Excel dataset (.xls, .xlsx).

Attribute	Required?	Notes
input	Y	Import records from this dataset.
output	Y	Create this focus as output. The focus must not already exist (unless **force** is `true`).
force	N	Boolean. If `true`, allow overwrite of existing output focus.
type	N	Interpret the input file as this type of third-party dataset.
fields	N	Import these fields only.
fieldsfile	N	Import only the fields listed in this file.
xfields	N	Import all fields except these.

Attribute	Required?	Notes
xfieldsfile	N	Import all fields except those listed in this file.
metadata	N	Create the metadata file from a source Excel dataset.

NOTES:

- If you attempt to import from an Excel dataset and the metadata file already exists, the operation fails. To avoid this, use the `-force` argument.

```
<qsimportstat input="cust.xls" output="id.ftr" xfields="CID"
force="true"/>

<qsimportstat input="DirectBank.xlsx" output="DirectBank_xlsx.ftr"
              metadata="DirectBank_xlsx.qsfm"/>
```

qsgenfdd

Create a flat file description (`.fdd` file) from data in a text file, unless a valid `.fdd` file already exists, in which case do nothing. The description is placed in the same directory as the source data, with a filename based on the source file name.

Attribute	Required?	Notes
input	Y	Generate a description for this text file.
template	N	Use the specified `.fdd` file as a template for the export format. Other options override or extend this format.
comment	N	Include this comment in any `.fdd` file generated.
dateformat	N	Write date fields using specified date format.
datemarker	N	Write date fields using this quoting character
defaultday	N	Use specified default day number.
defaultmonth	N	Use specified default month number.
headers	N	Boolean. If `true`, write an initial header record with field names.

Attribute	Required?	Notes
separator	N	Separate fields with this character
stringmarker	N	Enclose string values with this quoting character.
null	N	Use this null marker instead of blank.

Examples:

```
<qsgenfdd input="SomeData.csv"/>
```

qsimportfocus

Create a focus from an existing focus.

Attribute	Required?	Notes
input	Y	Read data from this focus.
output	Y	Create this focus as output (use **force** to overwrite existing focus).
force	N	Boolean. If `true`, allow overwrite of existing output.
fields	N	Import only this comma-separated list of fields.
fieldsfile	N	Import only the fields listed in this file.
xfields	N	Import all fields except these.
xfieldsfile	N	Import all fields except those listed in this file.
records	N	Import only the subset of records that satisfy this logical FDL expression.
recordsfile	N	Use the FDL expression in this file to select records.
subfocus	N	Read from this subfocus instead of the default.

Attribute	Required?	Notes
preservetypes	N	Preserve legacy datatypes from a focus created using an earlier version of Spectrum Miner, instead of converting such fields into fields with the standard integer, real, date, and string datatypes.

Examples:

```
<qsimportfocus input="cust_s.ftr" output="cust_copy.ftr" fields="CID,Age"
/>
```

Exporting data

qsdbcreatetable

Create a new database table (containing no records).

Attribute	Required?	Notes
udc	Y	Use these database credentials.
sql	Must specify sql or sqlfile or nested sql [see **Inline scripting** on page 36] element, or both table and focus	Read data from this SQL create table statement.
sqlfile		Read data from the SQL create table statement in the specified file.
focus		Create columns in the database table using field information from this focus.
table		Read data from the specified table.
output	N	Write the generated SQL create table statement to this file instead of executing it.
fields	N	Comma-separated list of fields to use from template focus.
fieldsfile	N	File listing fields to use from template focus. Only valid if table specified.

Attribute	Required?	Notes
xfields	N	Use all fields except these.
xfieldsfile	N	Use all fields except those listed in this file.
catalog	N	Create the database table in this catalog. Only valid if table specified.
schema	N	Create the database table in this schema. Only valid if table specified.
subfocus	N	Use this subfocus instead of the default.

NOTES:

- A nested sql element can be used to specify more complex SQL inline, rather than providing it within an attribute or via an external file. Remember to use `CDATA` or XML entities to escape special characters [see **Special characters in XML** on page 25].

Examples:

```
<qsdbcreatetable udc="TargetDB" table="SCORE_TABLE"
                 focus="template.ftr" fields="CustId,Score"/>
```

qsdbinsert

Insert data from a focus into an existing database table.

Attribute	Required?	Notes
input	Y	Source data
udc	Y	Database credentials to use.
table	Y	Insert into this existing database table.
fields	N	Comma-separated list of fields to insert
fieldsfile	N	File containing field list
xfields	N	Export all fields except these.

Attribute	Required?	Notes
xfieldsfile	N	Export all fields except those listed in this file.
catalog	N	Locate the database table in this catalog.
schema	N	Locate the database table in this schema
subfocus	N	Export this subfocus instead of the default.

Examples:

```
<qsdbinsert udc="ecstest@eigg" input="SRC_SCORE"
      table="SRC_SCORE" fields="CUST_ID,SCORE"/>
```

qsdbupdate

Modify (update) existing records in a database table with new values from a focus.

Attribute	Required?	Notes
input	Y	Read data from this focus
keys	Y	Comma-separated list of keys to match against table.[14]
udc	Y	Database credentials to use.
table	Y	Update data in this existing database table.
fields	N	Comma-separated list of fields to update
fieldsfile	N	File containing field list
xfields	N	Export all fields except these.
xfieldsfile	N	Export all fields except those listed in this file.

[14] There is also a keysfile attribute that corresponds to the `-key @file` command-line option.

Attribute	Required?	Notes
catalog	N	Locate the database table in this catalog.
schema	N	Locate the database table in this schema
subfocus	N	Export this subfocus instead of the default.

Examples:

```
<qsdbupdate memory="255" udc="ecstest@eigg"
            input="SRC_SCORE" table="SRC_SCORE" fields="SCORE"
            keys="CUSTOMER_NUMBER" />
```

qsexportflat

Export a focus to a flat file.

Attribute	Required?	Notes
input	Y	Read data from this focus
output	Y	Create or overwrite this flat file as output.
template	N	Use the specified `.fdd` file as a template for the export format. Other options override or extend this format.
fields	N	Comma-separated list of fields to export
fieldsfile	N	File containing field list
xfields	N	Export all fields except these.
xfieldsfile	N	Export all fields except those listed in this file.
records	N	Export only the subset of records that satisfy this logical FDL expression. Any fields used in the expression must be exported.
recordsfile	N	Specify a file containing a selection expression as above.

Attribute	Required?	Notes
alwaysquotestrings	N	Boolean. Quote all string values, instead of quoting only strings that contain special characters.
dateformat	N	Write date fields using specified date format.
datemarker	N	Write date fields using this quoting character
defaultday	N	Use specified default day number.
defaultmonth	N	Use specified default month number.
fdd	N	Boolean. If `true`, force creation of `.fdd` file (default only if needed).
fixedformat	N	Boolean. If `true`, write fixed-width instead of delimited output.
headers	N	Boolean. If `true`, write an initial header record with field names.
separator	N	Separate fields with this character
stringmarker	N	Enclose string values with this quoting character.
null	N	Use this null marker instead of blank.
subfocus	N	Export this subfocus instead of the default.

Examples:

```
<qsexportflat memory="255" input="cust_copy.ftr" output="cust_copy.fdd"
/>
```

qsexportstat

Export a focus to a dataset in a third-party format. See qsimportstat [see **qsimportstat** on page 40] for supported formats.

Attribute	Required?	Notes
input	Y	Export records from this focus.

Attribute	Required?	Notes
output	Y	Create this file as output.
subfocus	N	Use the specified subfocus of the source focus.
overwrite	N	Boolean. If `true`, allow overwrite of an existing output file. Without this option, if the dataset specified using output already exists, the task does nothing (except issue a warning).
type	N	Export the records using this third-party format (inferred from output file extension if not specified).
fields	N	Export these fields only.
fieldsfile	N	Export only the fields listed in this file.
xfields	N	Export all fields except these.
xfieldsfile	N	Export all fields except those listed in this file.
records	N	Export only records that satisfy this logical FDL expression.
recordsfile	N	Export only records that satisfy the logical FDL expression in this file.

Examples:

```
<qsexportstat input="cust.ftr" output="cust.sd2" type="xls"/>
```

Combining foci

qsjoin

Add one or more fields from one or more source foci to a target focus. The records are matched using one or more key fields and all foci must be sorted by these key fields. The number and order of records in the output focus is the same as in the primary focus. If the target focus is an original extract focus, you cannot overwrite it (unless you use force). Corresponding key fields must have compatible data types (that is, all numeric, all date or all string).

Attribute	Required?	Notes
input	Y	Location of base focus for join.
keys	Y	Comma-separated list of key fields.[15]
join	Either attribute or nested element required.	Comma-separated list of foci to join to input focus. Alternatively, use one or more nested **filesets** to join multiple foci, and/or nested `join` elements as described below.
output	N	Save the resulting focus to this location instead of back to the input focus (default). The output focus shares data with the input focus.
force		Boolean. If `true`, allow overwriting of existing output.
subfocus		Use the specified subfocus of the base focus. To specify subfoci from joined foci, use nested `join` elements.
fields	N	Comma-separated list of fields to select from join focus
fieldsfile	N	File containing field list as above
xfields	N	Join all fields except for this comma-separated list from the (single) secondary focus that is specified by the join attribute. To specify field inclusion or exclusion with multiple joined foci, use nested `join` elements.
xfieldsfile	N	Join all fields except for those listed (one per line) in this file from the (single) secondary focus that is specified by the join attribute. To specify field inclusion or exclusion with multiple joined foci, use nested `join` elements.
unmatched	N	Create a separate focus with this name or prefix that contains records from the joined focus that did not match any record from the input focus, instead of discarding them.
equalnulls	N	Boolean. Treat occurrences of the null value in key fields as equal to one another, instead of not equal to one another (or to any non-null value)
importmeta	N	Boolean. Import field metadata from the secondary foci.

[15] There is also a keysfile attribute that corresponds to the `-key @file` command-line option.

Attribute	Required?	Notes
match	N	For each secondary focus, create an additional integer field in the primary focus, named using this prefix, that contains the value 1 for those records from the primary focus that matched a record from the secondary focus, and the value 0 for those records that did not match.
onetoone	N	Boolean. Permit multiple records in the primary focus to share a given combination of key-field values, and attempt to join fields from successive records in each secondary focus that share the same combination of key-field values, in such a way that records are matched one-to-one.

NOTES:

- This task supports nested **filesets** to specify multiple secondary join foci.
- If detailed control over field inclusion/exclusion is required when multiple foci are joined, use nested `join` element for those datasets. Each contains a required focus attribute specifying the joined focus location, along with any one of the fields, fieldsfile, , or attributes described above.

Examples:

```
<qsjoin input="cust.ftr" join="cust_measure.ftr" keys="CID"
output="output.ftr"/>

<qsjoin input="cust.ftr" join="join.ftr" keys="CID" output="output.ftr"

        xfields="a,b,c"/>

<qsjoin input="cust.ftr" join="a.ftr,b.ftr,c.ftr" keys="CID"
output="join.ftr"/>

<qsjoin input="base.ftr" keys="a,b,c" output="joined.ftr">
  <join focus="b.ftr"/>
</qsjoin>

<qsjoin input="base.ftr" keys="a,b,c" output="joined.ftr">
  <join focus="a.ftr" subfocus="subfocus1" fields="a,b,c"/>
  <join focus="b.ftr"/>
  <fileset dir="data" includes="*.ftr"/>
  <join focus="c.ftr" xfieldsfile="not.lst"/>
</qsjoin>
```

qsmerge

Create a sorted focus by interleaving records from a sorted primary focus and one or more sorted secondary foci. Each field that is in the primary focus and also in one or more of the secondary foci must have exactly the same data type in each focus. The output focus has the same set of fields as the primary focus.

Attribute	Required?	Notes
input	Y	Location of base focus for merge.
output	Y	Create this focus as output, which must not already exist unless **force** is specified. The output focus shares no data with the input focus.
keys		Comma-separated list of key fields.[16]
merge	Either attribute or nested element required.	Comma-separated list of foci to merge with input focus. Alternatively, use one or more nested **filesets** to merge multiple foci.
force	N	Boolean. If `true`, allow overwriting of existing output.
subfocus	N	Use the specified subfocus of the base focus. To specify subfoci from merge foci, use nested `merge` elements.
nodups	N	Boolean. When multiple records in one or more of the base or merge input foci share a given combination of key-field values, include only the first record, according to an ordering that puts the base focus first, then the merge foci in the order they appear.
equalnulls	N	Boolean. Treat occurrences of the null value in key fields as equal to one another, instead of not equal to one another (or to any non-null value).

Examples:

```
<qsmerge input="base.ftr" output="merge.ftr" keys="CID" merge="one.ftr"
         equalnulls="true" nodups="false"/>
```

[16] There is also a keysfile attribute that corresponds to the `-key @file` command-line option.

```
<qsmerge input="cust1.ftr" merge="cust2.ftr,cust3.ftr,cust4.ftr"
         output="cust_merge.ftr" keys="CID"/>

<qsmerge input="base.ftr" output="merged.ftr" keys="ABC">
  <merge focus="cust.ftr" subfocus="subfocus1"/>
  <fileset dir="data" includes="*.ftr"/>
</qsmerge>
```

Enhancing foci

qsexportmetadata

Export metadata from the source focus, including focus history, binnings, comments, derivations, interpretations, record selections, subfocus structure, and default subfocus. By default, write the information to standard output.

Attribute	Required?	Notes
input	Y	Export metadata from this focus
output	N	Create this metadata file instead of writing to standard output.

Examples:

```
<qsexportmetadata input="cust.ftr" output="metadata.xml"/>
```

qsimportmetadata

Import metadata, including binnings, comments, derivations, interpretations, record selections, subfocus structure, and default subfocus from the metadata file to the source focus. If the focus is an original extract focus, you cannot overwrite it (unless you use **force**).

Attribute	Required?	Notes
input	Y	Add metadata to this focus

Attribute	Required?	Notes
metadata	Y	Read metadata from this XML file.
output	N	Location for the output focus (defaults to save back to input location).
force	N	Boolean. If `true`, allow overwriting of existing output.
details		Comma-separated list of kinds of metadata to import, which can include `binnings`, `comments`, `derivations`, `history`, `interpretations`, `selections`, `launch` (default subfocus), and `subfoci` (subfocus structure). Categorical interpretations are classified as binnings, not interpretations. In the absence of this attribute, all kinds of metadata except history are imported.
fields	N	Instead of importing field metadata from all fields, import metadata from just these fields.
fieldsfile	N	Instead of importing field metadata from all fields, import metadata from just the fields listed (one per line) in this fields file.
xfields	N	Import field metadata from all fields except these fields.
xfieldsfile	N	Import field metadata from all fields except the fields that are listed (one per line) in this fields file.
dryrun	N	Boolean. If `true`, do not import metadata, but display information about the metadata that would have been imported.
warn	N	Boolean. If `true`, rather than aborting the operation if import of some metadata fails, issue a warning, and attempt to import the remaining metadata.

NOTES:

- Imported comments may contain standard HTML markup (including hyperlinks), but to do so they must either be marked as **xhtml** and be enclosed within a `div` tag, or else must be fully entitized, must begin with `<html>`, and must end with `</html>`.
- If you import a subfocus structure, the newly created subfocus automatically inherits field attributes (such as analysis candidates and binnings) from the parent subfocus. This occurs even if you do not explicitly specify the attributes in the subfocus definition because the attributes are set in the process of applying the subfocus structure.

Examples:

```
<qsimportmetadata input="cust.ftr" output="cust_comment.ftr"
metadata="metadata.xml"/>

<qsimportmetadata input="DirectBank.ftr"
output="DirectBank_selection.ftr"
                  metadata="selection.qsfm" details="selections"/>

<qsimportmetadata input="cust.ftr" output="cust_comment.ftr">
  <metadata xmlns="http://www.quadstone.com/xml">
    <focus>
      <comment xhtml="true">
        <div>
         This focus was created for the <strong>Retail Analysis</strong>
 project<br/>
          An audit is available on the
          <a href="http://intranet.company.com/audits">intranet</a>.
        </div>
      </comment>
     <field name="Age"><comment>This is the age field</comment></field>

    </focus>
  </metadata>
</qsimportmetadata>
```

qsupdate

Copy the metadata from one focus to another.

Attribute	Required?	Notes
from	Y	Read metadata from this focus
to	Y	Apply metadata to this focus (must not be an original extract focus unless **force** is `true`).
force	N	Boolean. If `true`, allow overwriting of existing output.

Examples:

```
<qsupdate from="template.ftr" to="target.ftr"/>
```

Transforming foci

qsderive

Derive new fields in a focus.

Attribute	Required?	Notes
input	Y	Derive new fields from this focus.
derivations	Either attribute or nested element required.	Use derivations found in this file which may contain TML `create` statements or XML `field` elements.
output	N	Save output to this location instead of back to the input focus. The output shares data with the input focus.
force	N	Boolean. If `true`, allow overwriting of existing output.
fields	N	Derive these fields (comma-separated list of field names) only.
fieldsfile	N	Derive the fields listed in this file only.
xfields	N	Derive all fields except these (comma-separated list of field names).
xfieldsfile	N	Derive all fields except those listed in this file.
random	N	Use this seed for all derived fields (Note: all derivations will then use the same sequence of random numbers.)
subfocus	N	Use this subfocus instead of the default.
savexml	N	Boolean. In addition to deriving fields, write out the derivation expression(s) in TML and XML formats to files output.`tml` and output.`xml`.
warn	N	Boolean. If `true`, continue deriving fields instead of aborting after the first failure.

Attribute	Required?	Notes
macros	N	Read macros from this text or XML file. See also the nested macro [see **macro** on page 36] element. Corresponds to the `-macro @file` command-line option.

Examples:

- Using embedded derivations:

```
<qsderive input="cust.ftr" output="cust_deriv.ftr" random="124882">
  <!-- Note both random fields will be identical due to fixed seed -->

  <macro name="p1" value="one"/>
  <macro name="p2" value="two"/>
  <derivations deleteonexit="false">
    create random${p1}:=rndUniform();
    create random${p2}:=rndUniform();
  </derivations>
</qsderive>
```

- Using inline XML derivations:

```
<qsderive input="cust.ftr" output="cust_deriv.ftr" random="124882">
  <derivations xmlns="http://www.quadstone.com/xml">
    <field name="random" type="real">
      <fdl>rndUniform();</fdl>
    </field>
  </derivations>
</qsderive>
```

- Using derivations script:

```
<qsderive input="cust.ftr" output="scored.ftr" warn="true"
derivations="score.fdl"/>
```

qsmeasure

Compute aggregations on a focus and write them as fields in a new focus.

Attribute	Required?	Notes
input	Y	Compute aggregations on this focus.

Attribute	Required?	Notes
output	Y	Create this focus as output. The output does not share data with the input, and must not exist unless **force** is `true`.
keys	Y	Group records for aggregation using this comma-separated list of fields as composite key.[17]
aggregations	Either attribute or nested element required	Use the aggregations specified in this comma-separated list of files, each containing either TML create statements or XML syntax.
force	N	Boolean. If `true`, allow overwriting of existing output.
fields	N	Aggregate using only this comma-separated list of fields.
fieldsfile	N	Aggregate using only the fields listed in this file.
xfields	N	Aggregate using all fields except these.
xfieldsfile	N	Aggregate using all fields except those listed in this file.
library	N	Include FDL function or TML parameter definitions from this comma-separated list of files. See also the nested library [see **Inline scripting** on page 36] element.
macros	N	Read macros from this text or XML file. See also the nested macro [see **macro** on page 36] element. Corresponds to the `-macro @file` command-line option.
statistics	N	Calculate the statistics defined in this comma-separated list of files, containing either TML calculate statements or XML syntax. See also the nested statistics [see **Inline scripting** on page 36] element.

Examples:

- using nested library and aggregations

```
<qsmeasure input="cust.ftr" output="cust_measure.ftr" keys="CID"
           library="functions.tml">
  <macro name="p1" value="1"/>
  <macro name="p2" value="2"/>
```

[17] There is also a keysfile attribute that corresponds to the `-key @file` command-line option.

```
  <aggregations deleteonexit="false">
    create cnt${p1} := count();
    create cnt${p2} := count();
  </aggregations>
</qsmeasure>
```

- Using inline XML aggregations

```
<qsmeasure input="cust.ftr" output="cust_measure.ftr" keys="CID">
  <aggregations xmlns="http://www.quadstone.com/xml">
    <field name="cnt1" context="aggregation">
      <fdl>count()</fdl>
      <by>Gender</by>
      <where>Age > 20</where>
    </field>
  </aggregations>
</qsmeasure>
```

- using external TML files

```
<qsmeasure memory="255" input="cust.ftr" output="cust_measure.ftr"
keys="CID" aggregations="measure.tml" library="library.tml">
  <macro name="param1" value="1" />
  <macro name="param2" value="2" />
</qsmeasure>
```

NOTES:

- Currently only a single nested `library` element or **library** attribute (with a single file, not a comma-separated list) can be specified.
- Statistics are not currently supported

qstrack

Derive fields according to field definitions that typically involve state variables, using the key field to identify groups of records for tracking state, resetting all state variables at the start of a new group.

Attribute	Required?	Notes
input	Y	Compute derivations on this focus.
output	Y	Create this focus as output. The output does not share data with the input, and must not exist unless **force** is `true`.

Attribute	Required?	Notes
keys	Y	Group records using this field as key.[18]
trackers	Either attribute or nested element required	Derive fields using the field definitions in this comma-separated list of files, either containing TML state-tracking statements or XML syntax.
force	N	Boolean. If `true`, allow overwriting of existing output.
fields	N	Consider only the specified fields from the source focus (to optimize performance of the command by avoiding consideration of unused fields).
fieldsfile	N	Consider only fields from the source focus that are listed (one per line) in the fields file.
xfields	N	Consider all fields from the source focus, except the specified fields (to optimize performance of the command by avoiding consideration of unused fields).
xfieldsfile	N	Consider all fields from the source focus, except the fields that are listed (one per line) in the fields file.
library	N	Include FDL function definitions from this comma-separated list of files. Expressions in the trackers files may involve these functions. See also the nested library [see **Inline scripting** on page 36] element.
macros	N	Read macros from this text or XML file. See also the nested macro [see **macro** on page 36] element. Corresponds to the `-macro @file` command-line option.
statistics	N	Compute statistics for the input focus according to the definitions in this comma-separated list of files, containing either TML calculate statements or XML syntax. Expressions in the trackers files may refer to these statistics. See also the nested statistics [see **Inline scripting** on page 36] element.

Examples:

```
<qstrack input="trans.ftr" output="trans_tracked.ftr" keys="CID">
  <trackers deleteonexit="false"><![CDATA[
    create runbal := (
      state bal := 0;
```

[18] Note that only a single key field is allowed (and that the equivalent command-line option is -key not -keys). There is also a keysfile attribute that corresponds to the `-key @file` command-line option.

```
      bal := bal + Value;
    );
    ]]>
  </trackers>
</qstrack>

<qstrack input="trans.ftr" output="trans_tracked.ftr" keys="CID">
  <trackers xmlns="http://www.quadstone.com/xml">
    <field name="runbal">
      <fdl><![CDATA[state bal := 0; bal := bal + Value;]]></fdl>
    </field>
  </trackers>
</qstrack>
```

qsrenamefields

Rename the fields in a focus (across all subfoci, if they exist).

Attribute	Required?	Notes
input	Y	Rename fields in this focus.
map	Exactly one of these is required.	Either a comma-separated list of field name reassignments or the name of a standard mapping (currently only `QSCompliant` is supported). Renamings are applied in the order in which they are specified.
mapfile		A file containing a list of field name re-assignments, either in XML format, or as a text file containing lines of form OldName=NewName.
output	N	Write the resulting focus to this file instead of modifying the input focus. The focus must not already exist (unless you use **force** is `true`).
force	N	Boolean. If `true`, allow overwriting of existing output.
invert	N	Boolean. Apply the name mapping in reverse.
mapping	N	Write a record of the mapped field names to this file (in XML format).

Note that the top-level inline XML element is named map, not mappingset as in the analogous data-build command.

Examples:

```
<qsrenamefields input="cust.ftr" output="cust_rename.ftr"
                map="CID=CustID,Gender=Sex" mapping="map.xml"

<qsrenamefields input="c2.ftr" map="Age=Age2,Gender=G2"/>

<qsrenamefields input="c3.ftr">
  <map xmlns="http://www.quadstone.com/xml">
    <map>
      <name>Age</name>
      <alias>Age2</alias>
    </map>
    <map>
      <name>Gender</name>
      <alias>G2</alias>
    </map>
  </map>
</qsrenamefields>

<qsrenamefields input="strange.ftr" map="QSCompliant"/>
```

qsselect

Derive a new field in a focus and use it to apply a record selection.

Attribute	Required?	Notes
input	Y	Derive the field and apply the record selection to this focus.
selections	Either attribute or nested element required.	Use the named derivations file containing TML create statements or XML syntax. If the **selection** attribute is defined, derive the field with that name, otherwise the first field specified. The derivation must have logical type, and records with value 1 (`true`) are selected.
force	N	Boolean. If `true`, allow overwriting of existing output.
output	N	Save result focus to this location instead of back to the input. Must not exist unless **force** is `true`.
selection	N	Use the field with this name instead of the first field in the selections specification.

Attribute	Required?	Notes
macros	N	Read macros from this text or XML file. See also the nested macro [see **macro** on page 36] element. Corresponds to the `-macro @file` command-line option.
random	N	Use this seed for all derived fields (Note: all derivations will then use the same sequence of random numbers.)
subfocus	N	Use this subfocus instead of the default.
savexml	N	Boolean. In addition to deriving fields, write out the derivation expression(s) in TML and XML formats to files output.`tml` and output.`xml`.

Examples:

```
<qsselect input="foo.ftr" output="bar.ftr" selection="thisfield"
          selections="thisfile.fdl"/>

<qsselect input="foo.ftr">
  <selections>
    create HasAccount := HasChecking or HasSavings;
  </selections>
</qsselect>

<qsselect input="foo.ftr">
  <selections xmlns="http://www.quadstone.com/xml">
    <field name="HasAccount" context="selection">
      <fdl>HasChecking or HasSavings</fdl>
    </field>
  </selections>
</qsselect>
```

qssort

Read the records from a focus, sort them in ascending order and write the sorted records to a new focus, or check that a focus is correctly sorted.

Attribute	Required?	Notes
input	Y	Sort data from this focus

Attribute	Required?	Notes
keys	Y	Sort data using a composite key comprised of this comma-separated list of fields.[19]
output	Exactly one of these is required	Create this focus as output, which must not exist unless **force** is `true`.
check		Boolean. If `true`, check whether the focus is sorted, returning an exit status of 0 (sorted) or 1 (not sorted).
force	N	Boolean. If `true`, allow overwriting of existing output. Note it is not advisable to overwrite the input focus.
subfocus	N	Read from this subfocus instead of the default.

NOTES:

- Even when the source focus (or subfocus) contains only a selection of records, the destination focus contains all the records from the source focus.

Examples:

- Sort a focus:

```
<qssort input="cust.ftr" output="cust_s.ftr" keys="CID" />
```

- Check sortedness and set a property that could be used for conditional execution

```
<qssort input="cust.ftr" check="true" keys="CID"
resultproperty="sortedflag" failonerror="false" />
```

Managing foci

qscopy

Copy a focus. The new focus does not share data with the original focus.

[19] There is also a keysfile attribute that corresponds to the `-key @file` command-line option.

Attribute	Required?	Notes
from	Either attribute or nested element required.	Copy this focus, or set of foci identified by a nested **fileset** element.
todir	Exactly one of these is required.	Copy to this directory
to		Copy single focus to this location (not allowed with nested **fileset**).
force	N	Boolean. If `true`, allow overwriting of existing output.

NOTES:

- `qscopy` will not overwrite an existing target focus.
- `qscopy` can optionally accept a **fileset** nested element, in which case the **todir** attribute must be used, and the **from** attribute must not.
- `qscopy` can also accept a nested **mapper** element. A `mapper` can only be used with **fileset** and only one `mapper` can be provided.

Examples:

- Simple copy:

```
<qscopy from="cust.ftr" to="cust_c.ftr" />
```

- Temporary copy:

```
<qscopy from="cust.ftr" tmp="tmp_cust.ftr" />
```

- Copy directory structure flattening it into a single directory

```
<qscopy todir="mapper_flatten" failonerror="true">
  <fileset dir="${data}" includes="**/*.ftr" />
  <mapper type="flatten" />
</qscopy>
```

- Copy directory structure and maintain directory hierarchy while renaming resultant foci.

```
<qscopy todir="mapper_copy" failonerror="true">
  <fileset dir="${data}" includes="**/*.ftr" />
  <mapper type="glob" from="*.ftr" to="*_copy.ftr" />
</qscopy>
```

qslink

Save a focus under a new name. The new focus shares data with the original focus.

Attribute	Required?	Notes
from	Either attribute or nested element required.	Link from this focus, or set of foci specified by a nested **fileset** element.
todir	Exactly one of these is required.	Link foci to copies in this directory.
to		Link single source focus to this location (not allowed with a nested **fileset**).
force	N	Boolean. If `true`, allow overwriting of existing output.

Examples:

- Simple link:

```
<qslink from="cust.ftr" to="cust_c.ftr" />
```

- Temporary link:

```
<qslink from="cust.ftr" tmp="tmp_cust.ftr" />
```

- Link from complex directory structure flattening it into a single directory

```
<qslink todir="mapper_flatten" failonerror="true">
  <fileset dir="${data}" includes="**/*.ftr" />
  <mapper type="flatten" />
</qslink>
```

- Link from complex directory structure and maintain directory hierarchy while renaming resultant foci.

```
<qslink todir="mapper_rename" failonerror="true">
  <fileset dir="${data}" includes="**/*.ftr" />
  <mapper type="glob" from="*.ftr" to="*_copy.ftr" />
</qslink>
```

qsmove

Move or rename a focus.

Attribute	Required?	Notes
from	Either attribute or nested element required.	Move this focus, or set of foci specified by a nested **fileset** element.
todir	Exactly one of these is required.	Move foci to this directory
to		Move single focus to this location (not allowed with a nested **fileset**).
force	N	Boolean. If `true`, allow overwriting of existing output.

NOTES:

- `qsmove` will not overwrite a focus unless **force** is `true`.
- `qsmove` can optionally accept a **fileset** nested element, in which case the **todir** attribute must be used, and the **from** attribute must not.
- `qsmove` can also accept a nested **mapper** element. A `mapper` can only be used with **fileset** and only one `mapper` can be provided.

Examples:

- Simple move

```
<qsmove from="cust.ftr" to="cust_moved.ftr" />
```

- Temporary move

```
<qsmove from="cust.ftr" tmp="tmp_cust_moved.ftr" />
```

- Move directory structure flattening it into a single directory

```
<qsmove todir="mapper_flatten" force="true" failonerror="true">
  <fileset dir="${data}" includes="**/*.ftr" />
  <mapper type="flatten" />
</qsmove>
```

- Move Directory structure and maintain directory hierarchy while renaming resultant foci.

```
<qsmove todir="mapper_copy" failonerror="true">
  <fileset dir="${data}" includes="**/*.ftr" />
  <mapper type="glob" from="*.ftr" to="*_moved.ftr" />
</qsmove>
```

qsremove

Delete one or more focus files, taking into account possible data dependencies.

Attribute	Required?	Notes
focus	Either attribute or nested element required.	Remove this focus. Alternatively, provide a nested **fileset** to remove several foci at once.
silent	N	Boolean. If `true`, print no output while removing (and don't warn if focus doesn't exist). Note there is no corresponding command-line option.
force	N	Boolean. If `true`, ignore any errors caused by failure to update links with related foci.
recursive	N	Boolean. If `true`, delete the focus and all dependent foci.

NOTES:

- If there is a circular dependency in the chain, or the parent of the focus chain is not writeable, an error will be displayed. In this case, you need to use both **force** and **recursive** to delete the focus.
- Removing a focus with qsremove will fail if you do not have write permission to the `.ftr` focus file.

Examples:

- Delete single focus

```
<qsremove focus="cust.ftr" silent="true"/>
```

- Delete foci based on **filesets**

```
<qsremove dir="${data}" silent="true">
```

```
  <fileset dir="${data}"
      includes="copydir/**/*.ftr, movedir/**/*.ftr, subdir/**/*.ftr"
/>
</qsremove>
```

qsarchive

Create an archive file from a list of foci or folders.

Attribute	Required?	Notes
input	Either attribute or nested element required.	Include this file or folder in the archive file. Alternatively, use one or more nested filesets to archive multiple foci.
output	Y	Create this archive file as output.

NOTES:

- If you create an archive file from a folder that contains foci that link to data outside the folder, the external data is copied into the archive.

Examples:

```
<qsarchive input="focus.ftr" output="focus.zip" />

<qsarchive input="resultsFolder" output="folder.zip" />

<qsarchive output="files.zip">
  <fileset dir=".">
    <include name="file*.ftr"/>
  </fileset>
</qsarchive>
```

qsunzip

Extract files and folders from an archive file.

Attribute	Required?	Notes
input	Y	Extract from this archive file.
output	N	Extract to an alternative directory to the one in which the archive file is located.
overwrite	N	Boolean. If `true`, overwrite any files or folders in the same location as the archive file.

NOTES:

- If you don't use the **overwrite** option and extract to a location that contains the same source files and folders, the extract will fail.

Examples:

```
<qsunzip input="firstquarter.zip" output="unzipped" />
```

Reporting tasks

qsaudit

Generate a report about a focus, or based on an existing report.

Attribute	Required?	Notes
input	Y	Generate a report based on this focus or report file.
output	Y	Write the resulting report to this file. This should be a `.qshtml` file when generating a full report, and a `.qsxml` file when generating XML.
fields	N	Report only on this comma-separated list of fields.
fieldsfile	N	Report only on the fields listed in this file.
xfields	N	Report on all fields except these.

Attribute	Required?	Notes
xfieldsfile	N	Report on all fields except those listed in this file.
records	N	Report only on records that satisfy this logical FDL expression.
recordsfile	N	Report only on records that satisfy the logical FDL expression in this file.
targets	N	Report on each of these comma-separated target fields, ignoring any objective interpretation in the input.
targetsfile	N	Report on each of the target fields listed in this file, ignoring any objective interpretation in the input.
notarget	N	Boolean. If `true`, ignore the objective interpretation in the input focus.
generate	N	Choose `Full` to generate a full report from an input focus, `HTML` to generate a full report from an existing input report, or `XML` to generate a partial report from an input focus. A partial report file contains enough information to form the basis for generating other reports, but is not itself viewable. The default is `Full`.
overwrite	N	Boolean. If `true`, overwrite any existing report of the same name.
paginate	N	Boolean. If `true`, place details of each field in a separate web page when generating a full report.
reference	N	Include differences from this report in the output. This option can only be used when the input is a focus.
htmlimages	N	Choose `largepng` to include images in high- and low-resolution bitmap and SVG formats; `smallpng` for low-resolution bitmap and SVG formats only; `svg` for SVG format only; or `none` to include no images at all. The default is smallpng.
subfocus	N	Report on this subfocus of the input focus instead of the default.
partitionfield	N	Use the specificed field as the partition field.
nopartition	N	Boolean. If `true`, create a non-uplift Profile and Audit, ignoring any partition interpretation in the focus.

NOTES:

- To unpack `.qshtml` and `.qsxml` archives, for viewing or reuse outside Spectrum Miner, use qshtmlunpack [see **qshtmlunpack** on page 74].

Examples:

```
<qsaudit input="monthly_build.ftr" overwrite="true"
targets="Spend,Frequency"/>
```

qsdtsnapshot

Generate a report about a decision tree model stored in a `.qsdt` file, or based on an existing report.

Attribute	Required?	Notes
input	Y	Generate the report on this model or report file.
output	Y	Write the resulting report to this file. This should be a `.qshtml` file when generating a full report, and a `.qsxml` file when generating XML.
audit	N	Choose `modeled`, `all`, or `none` to include an audit of only the fields used by the model (default), all of the fields in the focus, or no fields.
descriptionfile	N	Include the contents of this file as a description in the report. (Corresponds to the `-description` command-line option.)
focus	N	Report on the application of the model to this focus, instead of to the focus on which the model was built.
generate	N	Choose `Full` to generate a full report from an input focus, `HTML` to generate a full report from an existing input report, or `XML` to generate a partial report from an input focus. A partial report file contains enough information to form the basis for generating other reports, but is not itself viewable. The default is `Full`.
overwrite	N	Boolean. If `true`, overwrite any existing report of the same name.
htmlimages	N	Choose `largepng` to include images in high- and low-resolution bitmap and SVG formats; `smallpng` for low-resolution bitmap and SVG formats only; `svg` for SVG format only; or `none` to include no images at all. The default is smallpng.
subfocus	N	Report on this subfocus of the input focus instead of the default.

NOTES:

- To unpack `.qshtml` and `.qsxml` archives, for viewing or reuse outside Spectrum Miner, use qshtmlunpack [see **qshtmlunpack** on page 74].

Examples:

```
<qsdtsnapshot input="tree.qsdt" output="cust_audit.qshtml"
focus="customer.ftr"/>
```

qsscsnapshot

Generate a report about a score card model stored in a `.qssc` file, or based on an existing report.

Attribute	Required?	Notes
input	Y	Generate the report on this model or report file.
output	Y	Write the resulting report to this file. This should be a `.qshtml` file when generating a full report, and a `.qsxml` file when generating XML.
audit	N	Choose `modeled`, `all`, or `none` to include an audit of only the fields used by the model (default), all of the fields in the focus, or no fields.
descriptionfile	N	Include the contents of this file as a description in the report. (Corresponds to the `-description` command-line option.)
focus	N	Report on the application of the model to this focus, instead of to the focus on which the model was built.
generate	N	Choose `Full` to generate a full report from an input focus, `HTML` to generate a full report from an existing input report, or `XML` to generate a partial report from an input focus. A partial report file contains enough information to form the basis for generating other reports, but is not itself viewable. The default is `Full`.
overwrite	N	Boolean. If `true`, overwrite any existing report of the same name.
htmlimages	N	Choose `largepng` to include images in high- and low-resolution bitmap and SVG formats; `smallpng` for low-resolution bitmap and SVG formats only; `svg` for SVG format only; or `none` to include no images at all. The default is smallpng.

Attribute	Required?	Notes
subfocus	N	Report on this subfocus of the input focus instead of the default.

NOTES:

- To unpack `.qshtml` and `.qsxml` archives, for viewing or reuse outside Spectrum Miner, use qshtmlunpack [see **qshtmlunpack** on page 74].

Examples:

```
<qsscsnapshot input="card.qssc" output="cust_audit.qshtml" audit="all"/>
```

qsdescribe

Show information about a focus.

Attribute	Required?	Notes
input	Y	Show metadata details for this focus.
fields	N	Boolean. If `true`, print details for each field in the focus.
detail	N	Boolean. If `true`, print additional detail for fields.
output	N	Write output to this file (default console).
subfocus	N	Show metadata details for only this subfocus, instead of the root focus and all subfoci contained within it.

Examples:

```
<qsdescribe input="monthly_build.ftr" output="monthly_build_stats.txt"
fields="true"/>
```

qsdescribestat

Display a list of the field names in a third-party source dataset, in plain-text format. See qsimportstat [see **qsimportstat** on page 40] for supported formats.

Attribute	Required?	Notes
input	Y	Show information about this dataset.
output	N	Write information to this file instead of to standard output.
detail	N	Boolean. List the datatypes of fields alongside their names.
type	N	Interpret the file as this third-party format (inferred from **output** file extension if not specified).

Examples:

<qsdescribestat input="cust.sd2" detail="true"/>

qshtmlunpack

Unpack a Spectrum Miner-generated report in the form of a `.qshtml` or `.qsxml` archive, creating the specified output directory to contain the components of the report. (Equivalent to the right-click option **Unpack to Folder** in Spectrum Miner.) In the case of an archived HTML report, the HTML file itself (within the output directory) is by default called `qsreport.html`.

Attribute	Required?	Notes
input	Y	Unpack this report file.
outputdir	Y	Create, and unpack to, this directory.
newhtmlfilename	N	Use the specified filename for the HTML file in an unpacked HTML report, instead of the default `qsreport.html`. (When unpacking an archived XML report, `qshtmlunpack` ignores this attribute, if present.)

Examples:

```
<qshtmlunpack input="cust_audit.qshtml" outputdir="results/cust_audit"/>
```

qsinfo

Display information on the locations and sizes of the files that constitute the source focus, including any data files shared with other foci. Display the source focus's relationships to other foci (listing

both foci that depend on it and foci on which it depends). Equivalent to the **Focus>Properties** option in Spectrum Miner.

Attribute	Required?	Notes
input	Y	The input focus to report on.
output	N	Write the report to this file, instead of standard output.
format	N	The format of the output report, either `HTML` or `XML`. The default is `XML`.

Examples:

```
<qsinfo input="cust.ftr" output="cust_info.html"/>
<qsinfo input="cust.ftr" output="cust_info.xml" format="xml"/>
```

qsxt

Generate an XML crosstab by applying a crosstab specification (exported from Spectrum Miner) to a focus.

Attribute	Required?	Notes
focus	Y	Generate a crosstab from this focus.
spec	Y	Use the crosstab specification in this file.
descriptionfile	N	Include the contents of this text file as a description of the crosstab. (Corresponds to the `-description` command-line option.)
subfocus	N	Apply the crosstab specification to this subfocus instead of the default.
output	N	Write the resulting crosstab to this file (rather than standard output).
comparable	N	Use the bins from the crosstab specification instead of from the focus.

Examples:

```
<qsxt focus="monthly_build.ftr" spec="monthly_report_template.qsxt" out-
put="report.qsxt"/>
```

Building models

qsdecisiontree

Use a decision-tree build specification to build a decision tree on the specified focus, and create a new decision-tree build report (containing decision-tree statistics, a decision-tree prediction score, and a calibration crosstab) in an XML-based file format.

Attribute	Required?	Notes
input	Y	Generate a decision tree from this focus.
build	Y	Use the decision-tree build specification in this file.
result	Y	Generate this XML decision-tree build report.
subfocus	N	Use the specified subfocus of the focus.
output	N	Create this focus, containing an additional derived field that applies the new decision-tree prediction score to the input focus.
force	N	Boolean. If `true` allow overwrite of existing output focus.

Examples:

```
<qsdecisiontree build="decisiontree-specification.xml"
input="CustApril.ftr"
                result="decisiontree-report.xml"
output="CustAprilPredictedAge.ftr"
```

qsscorecard

Use a scorecard build specification to build a scorecard on the specified focus, and create a new scorecard build report (containing scorecard statistics, a scorecard prediction score, and a calibration crosstab) in an XML-based file format.

Attribute	Required?	Notes
input	Y	Generate a scorecard from this focus.
build	Y	Use the scorecard build specification in this file.
result	Y	Generate this XML scorecard build report.
subfocus	N	Use the specified subfocus of the focus.
output	N	Create this focus, containing an additional derived field that applies the new scorecard prediction score to the input focus.
force	N	Boolean. If `true` allow overwrite of existing output focus.

NOTES:

- For a binary objective, **qsscorecard** uses logistic regression; for a continuous objective, it uses linear regression.

Examples:

```
<qsscorecard build="scorecard-specification.xml" input="CustApril.ftr"

             result="scorecard-report.xml"
output="CustAprilPredictedAge.ftr" />
```

Other tasks

Note that these tasks do not support the normal common attributes [see **Common attributes and shortcuts** on page 33].

antimport

This is simply a synonym for Ant's **import** task, which imports another Ant build file into the current build script, but limited to pure Ant syntax and none of the `qsbuild` extensions.

Attribute	Required?	Notes
file	Y	The file to import. If this is a relative filename, the filename will be resolved relative to the importing file.
optional	N	Boolean. If true, do not stop the build if the file does not exist, default is false.

Example:

```
<antimport file="common-targets.xml" />
```

With imported file of the form:

```
<?xml version="1.0" ?>
<project name="imported" basedir="." default="echotask">
<!-- only standard ant tasks
  - not things specific to qsbuild
  e.g. foreach, dependency, qscall -->
<target name="echotask">
  <echo message="${message}"/>
</target>
<target name="anotherechotask">
  <antcall target="echotask">
    <param name="message" value="${message}"/>
  </antcall>
</target>
</project>
```

defaults

This task simply contains default values for one or more arbitrary attributes, for example, `<defaults key="CustId" memory="512"/>`. Subsequent task elements will inherit any unspecified legal attribute from the most recently specified default value in a parent element[20]. This task can also

[20] Note this means that targets pick up defaults where they are defined, not where they are referenced via `runtarget` or `dependency`.

appear at the top-level inside the `build` element (outside any enclosing `target`) to specify defaults for all targets.

parameter

This task is used to declare a user-configurable property, optionally specifying its default value.

Attribute	Required?	Notes
name	Y	Name of property to set
value	N	Optional default value for property
type	N	Required type, from the set: `string`, `file`, `directory`, `real`, `integer`, `date`.
format	N	Required layout for date properties (see qsdateproperty [see **qsdateproperty** on page 80]).

Examples:

```
<parameter name="month" value="April" type="date" format="%B"/>
```

NOTES:

- The `parameter` element can only appear at the top level of a build plan (inside the `build` element but outside any enclosing `target`).
- If **value** is not specified, the corresponding property will be unset during the build unless the user defines a value at runtime.
- Both **type** and **format** are used only by the graphical user interface to provide appropriate value selectors and input validation.

qscall

This is simply a synonym for Ant's **antcall** task, which allows a target to be called in a new build context, potentially with locally changed property values. Note that no property values are returned by `qscall`, in contrast to **runtarget** which executes a target in the existing build context (with any new property settings reflected in the caller).

Examples:

```
<qscall target="subroutine">
  <param name="something" value="Testing 1...2..3..."/>
</qscall>

...

<target name="subroutine">
  <echo message="Parameter something = ${something}"/>
</target>
```

qsdateproperty

This task supports date parsing, reformatting and arithmetic from string properties, with a syntax based on Ant's **propertyfile** task.

Attribute	Required?	Notes
property	Y	Property to write output date/time value to.
unit	N	Time unit from the set: `millisecond`, `second`, `minute`, `hour`, `day`, `week`, `month`, `year`. Defaults to `day`.
inputpattern	N	See below, defaults to `%Y-%m-%d %H:%M:%S`
outputpattern	N	See below, defaults to **inputpattern** if not set.
value	N	Date/time value to parse using **inputpattern**. Defaults to current local time.
offset	N	Numeric amount of units to add to input date.

The attributes inputpattern and outputpattern define the format of the date string as it is input and output from the source and target properties. The pattern is a format string using the special codes from the table below to describe components of the date format. By convention the %-escapes are used, although the Java date format strings are also valid: thus if a date format needs to contain A–Z or a–z characters directly, they must be quoted with single quotes. Other characters can be embedded directly. For example:

```
<qsdateproperty property="somevalue" outputformat="%H%M'hours'"/
```

Format	Java date format	Description
%%	%	The percent sign itself: `%`
%a	EEE	Abbreviated day name (Mon, Tue, . . . , Sun)
%A	EEEE	Full day name (Monday, Tuesday, . . . , Sunday)
%b	MMM	Abbreviated month name (Jan, Feb, . . . , Dec)
%B	MMMM	Full month name (January, February, . . . , December)
%c	EEE MMM d HH:mm:ss yyyy	Equivalent to `%a %b %e %T %Y`
%C	EEE MMM d HH:mm:ss zzz yyyy	Same as `%a %b %e %T %Z %Y`
%d	dd	Two-digit day of month (01, 02, . . . , 31)
%D	MM/dd/yy	Equivalent to `%m/%d/%y`
%e	d	(Not quite) Space-padded day of month (1, 2, . . . , 31)
%h	MMM	Abbreviated month name (Jan, Feb, . . . , Dec)
%H	HH	24-hour hour of the day (00, 02, . . . , 23)
%I	hh	12-hour hour of the day (01, 02, . . . , 12)
%j	DDD	Julian 3-digit day of the year (001, 002, . . . , 366)
%m	MM	2-digit month number (01, 02, . . . , 12)
%M	mm	2-digit minute of the hour (00, 01, . . . , 59)
%p	a	AM or PM, according to the time of day
%r	hh:mm:ss a	Equivalent to `%I:%M:%S %p`
%R	HH:mm	Equivalent to `%H:%M`
%S	ss	2-digit second of the minute (00, 01, . . . , 61)

Format	Java date format	Description
%T	HH:mm:ss	Equivalent to `%H:%M:%S`
%y	yy	2-digit year of the century (00, 01, . . . , 99)
%Y	yyyy	The year, using four digits
%Z	zzz	The name of the current locale's time zone

The Java Date Formats are detailed below:

Symbol	Meaning	Presentation	Example
G	era designator	(Text)	AD
y	year	(Number)	1996
M	month in year	(Text & Number)	July & 07
d	day in month	(Number)	10
h	hour in am/pm (1–12)	(Number)	12
H	hour in day (0–23)	(Number)	0
m	minute in hour	(Number)	30
s	second in minute	(Number)	55
S	millisecond	(Number)	978
E	day in week	(Text)	Tuesday
D	day in year	(Number)	189
F	day of week in month	(Number)	2 (2nd Wed in July)
w	week in year	(Number)	27

Symbol	Meaning	Presentation	Example
W	week in month	(Number)	2
a	am/pm marker	(Text)	PM
k	hour in day (1–24)	(Number)	24
K	hour in am/pm (0–11)	(Number)	0
Z	time zone	(Text)	Pacific Standard Time
' (single quote)	escape for text	(Delimiter)	
'' (two single quotes)	single quote	(Literal)	'

The example code below:

```
<qsdateproperty property="out1"/>
<echo message="Current Time = ${out1}"/>
<sleep seconds="3"/>
<qsdateproperty property="out2" outputpattern="%Y/%m/%d %H:%M:%S"/>
<echo message="Current time is now = ${out2}"/>
<qsdateproperty property="out3" value="${out2}" inputpattern="%Y/%m/%d
 H:%M:%S" outputpattern="%Y.%m.%d %H:%M:%S" offset="1" unit="hour"/>
<echo message="Current time +1 hour = ${out3}"/>
<qsdateproperty property="out4" value="${out2}" inputpattern="%Y/%m/%d
 H:%M:%S" outputpattern="%Y.%m.%d %H:%M:%S" offset="-1" unit="hour"/>
<echo message="Current time -1 hour = ${out4}"/>
<qsdateproperty property="out5" value="${out2}" inputpattern="%Y/%m/%d
 %T" outputpattern="%c" offset="-1" unit="day"/>
<echo message="Current time -1 day = ${out5}"/>
<qsdateproperty property="out6" value="${out2}" inputpattern="%Y/%m/%d
 %T" outputpattern="%C" offset="-1" unit="month"/>
<echo message="Current time -1 month = ${out6}"/>
<sleep seconds="2"/>
<qsdateproperty property="out7" outputpattern="%D %T" offset="10"
unit="week"/>
<echo message="New time +10 weeks = ${out7}"/>
<qsdateproperty property="out8" value="${out7}" inputpattern="%D %T"
outputpattern="%D %T" offset="1" unit="year"/>
<echo message="New time +1 year = ${out8}"/>
<qsdateproperty property="out9" outputpattern="%T %p %Z"/>
<echo message="Current time is = ${out9}"/>
<qsdateproperty property="out10" outputpattern="%r"/>
<echo message="Current time is = ${out10}"/>
```

```
<qsdateproperty property="out11" inputpattern="%C" value="${out6}"
offset="10" unit="year"/>
<echo message="${out6} + 10 years = ${out11}"/>
```

produces output similar to the following:

```
[echo] Current Time = 2003-12-30 10:45:16
[echo] Current time is now = 2003/12/30 10:45:19
[echo] Current time +1 hour = 2003.12.30 11:45:19
[echo] Current time -1 hour = 2003.12.30 09:45:19
[echo] Current time -1 day = Mon Dec 29 10:45:19 2003
[echo] Current time -1 month = Sun Nov 30 10:45:19 EST 2003
[echo] New time +10 weeks = 03/09/04 10:45:21
[echo] New time +1 year = 03/09/05 10:45:21
[echo] Current time is = 10:45:21 AM EST
[echo] Current time is = 10:45:21 AM
[echo] Sun Nov 30 10:45:19 EST 2003 + 10 years = 2013-11-30 10:45:19
```

qsmapgen

From one or more categorical hierarchy files (with filename extension `.hrc`), create a representation of the categorical hierarchies suitable for use in the Decisionhouse Map Viewer. Create the map files in the specified directory, using the specified map name as the basis for filenames.

Attribute	Required?	Notes
input	Y	Generate a map from the categorical hierarchy information in this `.hrc` file (or comma-separated list of files).
name	Y	Create a map with this name.
output	Y	Create the map files in this directory.
overwrite	N	Boolean. Overwrite any existing map files of the same name.
style	N	Specify either `radial_layers` (default) or `radial_drill`, to generate maps as either layered or drilldown respectively. A layered map comprises a separate map file for each level in the hierarchy; a drilldown map comprises a base-level map plus a hierarchy of drilldown maps.

Examples:

```
<qsmapgen input="foo.hrc" name="somemap"/>
<qsmapgen input="a.hrc,b.hrc,c.hrc" name="linked" output="somedir"
overwrite="true" style="radial_drill" />
```

5 - Standard and third-party tasks

In this section

Standard Ant tasks 87
Third-party tasks 90

Standard Ant tasks

For further details and examples, see the **Overview of Ant Tasks** or follow links from individual tasks. Note however that in contrast to Ant, the Data Build Manager uses case-sensitive validation: all task and attributes names must appear entirely in lowercase.

Property tasks

available Sets a property if a specified file, directory, class in the classpath, or JVM system resource is available at runtime.

basename Sets a property to the last element of a specified path.

buildnumber Task that can be used to track build numbers.

condition Sets a property if a certain condition holds true—this is a generalization of **available** and **uptodate**.

dirname Sets a property to the value of the specified file up to, but not including, the last path element.

echoproperties Lists the current properties.

loadfile Loads a file into a property.

loadproperties Set multiple Ant properties from a Java property file: a text file containing name=`value` records.

pathconvert Converts a nested path, path reference, filelist reference, or **fileset** reference to the form usable on a specified platform and/or to a list of items separated by the specified separator and stores the result in the specified property.

property Sets a property (by name and value), or set of properties (from a file or resource) in the build.

propertyfile Creates or modifies property files.

tstamp Sets the DSTAMP, TSTAMP, and TODAY properties in the current build, based on the current date and time.

uptodate Sets a property if a given target file is newer than a set of source files.

xmlproperty Loads property values from a valid XML file.

Archive tasks

gunzip Expands a GZip file.

gzip GZips a set of files.

tar Creates a tar archive.

untar Untars a tarfile.

unzip Unzips a zipfile.

zip Creates a zipfile.

File tasks

checksum Generates a checksum for a file or set of files. This task can also be used to perform checksum verifications.

chmod Changes the permissions of a file or all files inside the specified directories.

concat Concatenates multiple files into a single one, or in to Ant's logging system.

copy Copies a file or **fileset** to a new file or directory.

delete Deletes either a single file, all files and sub-directories in a specified directory, or a set of files specified by one or more **fileset**s.

fixcrlf Modifies a file to add or remove tabs, carriage returns, linefeeds, and EOF characters.

mkdir Creates a directory. Non-existent parent directories are created, when necessary.

move Moves a file to a new file or directory, or a set(s) of file(s) to a new directory.

replace A directory-based task for replacing the occurrence of a given string with another string in selected file.

replaceregexp A directory-based task for replacing the occurrence of a given regular expression with a substitution pattern in a file or set of files.

tempfile Generates a name for a new temporary file and sets the specified property to that name.

touch Changes the modification time of a file and possibly creates it at the same time.

Execution tasks

apply Executes a system command for each of a nested list of files.

exec Executes a system command.

java Executes a Java class within the running (Ant) VM, or in another VM.

sleep Suspends execution for a fixed period, giving a delay between tasks.

Other useful tasks

echo Echoes text to `System.out` or to a file.

fail Exits the current build optionally printing additional information.

input Allows user interaction during the build process by displaying a message and reading a line of input from standard input.

mail A task to send SMTP e-mail. For example:

```
<target name="sendresults">

<if><isset property="_fail_message"/>
  <then><property name="subject" value="Build failed!"/></then>
    <else><property name="subject" value="Success!"/></else>
</if>

<mail mailhost="smtp.somewhere.com" mailport="25" subject="${subject}"

    from="Joe.Shmoe@somewhere.com" tolist="Jane.Doe@elsewhere.com"
files="${logfile">
  <message>Here's the build log from ${logfile}.
  The value of _fail_message is:
  ${_fail_message}</message>
</mail>

</target>
```

record Runs a listener that records the logging output of the build-process events to a file. Several recorders can exist at the same time.

script Executes an external script in one of several BSF-supported scripting languages.

sql Executes a series of SQL statements via JDBC to a database.

xmlvalidate Checks that XML files are valid (or only well-formed).

xslt/style Processes a set of documents via XSLT.

Remote tasks

These tasks may require additional setup beyond the standard Spectrum Miner configuration. Contact **software.support@pb.com** for details.

ftp Implements a basic FTP client that can send, receive, list, and delete files, and create directories.

get Gets a file from a URL.

telnet Task to automate a remote telnet session. This task uses nested read and write tags to indicate strings to wait for and specify text to send.

Third-party tasks

The Data Build Manager exposes several third-party tasks from the **Ant-Contrib project**. For more details and examples, see the **Overview of Ant-Contrib Tasks**, or follow links from the individual task summaries below.

Logic tasks

foreach The foreach task iterates over a list, a list of paths, or both, calling a parameterized target for each item. These iterations can also happen in parallel (see Section **Concurrency** on page 24).

Note that the **inheritall** attribute defaults to `false` here (in contrast to qscall [see **qscall** on page 79]) so that by default properties are *not* propagated from the caller. This is a common source of confusion.

The Data Build Manager extends this task so that it can optionally substitute a nested `tasks` element in place of the **target** attribute. The `tasks` element is a container for an arbitrary sequence of tasks, acting as an anonymous target to be iterated upon. For example:

```
<foreach list="1,2,3" param="idx" parallel="true" maxthreads="2">
  <tasks>
    <echo message="idx = ${idx}"/>
    <sleep seconds="2" concurrent="true"/>
    <sleep seconds="1" concurrent="true"/>
    <echo message="slept for 2 & 1 seconds concurrently"/>
  </tasks>
</foreach>
```

Here's an example using `foreach` with a nested `tasks` element and also using a nested `path` element to specify a list of files via wildcards, instead of an explicit list of files. This is a very useful idiom for processing batches of files in a flexible and robust way:

```
<foreach param="cdr_focus">
  <!-- loop over all files matching this pattern -->
  <path><fileset dir="." includes="CDR2005*_sort.ftr"/></path>
  <tasks> <!-- do these steps with cdr_focus set to each file in turn
-->
    <!-- get the stem of the file to make a corresponding output -->
    <basename property="basename" file="${cdr_focus}" suffix=".ftr"/>
    <qsmeasure keys="ORIGIN" input="${cdr_focus}"
output="${basename}_minmax.ftr">
      <aggregations>
create firstCall := min(OCCURRED);
create lastCall := max(OCCURRED);
      </aggregations>
    </qsmeasure>
  </tasks>
</foreach>
```

if Provides an `if/then/elseif/else` container for other tasks.

switch Choose which task(s) to execute based on the value of a property.

runtarget Call another target in the same build context, so that the called target sees (and sets) the same properties as the caller. Compare qscall [see **qscall** on page 79].

Other tasks

shellscript This task allows the user to execute an embedded script against a particular shell program on a machine, extending the `exec` task.

stopwatch Start/stop multiple stopwatches to measure performance characteristics of a build

propertycopy Copies the value of a named property (possibly including property substitution) to another property. This is useful for setting a property to the value of another property, chosen dynamically based on the value of one or more other properties.

For example: `<propertycopy property="db.user" from="db.${database}.user"/>`.

Appendix

In this section

Appendix 93

A - Appendix

In this section

Credits 94

Credits

This product uses software developed by the **Apache Software Foundation**, including:

- **Apache Ant** (distributed under the **Apache Software License**)
- **Ant-Contrib Tasks** (distributed under the **Apache Software License**

This product uses "The Saxon XSLT Processor from Michael Kay", freely available from **http://saxon.sourceforge.net/** and distributed under the **Mozilla Public License**.

This product includes **Jing & Trang** © 2001–2003 Thai Open Source Software Center Ltd. All rights reserved. Distributed under the **conditions of the Thai Open Source Software Center Ltd.**.

This product uses Jython, available from **http://www.jython.org/**, and distributed under the **JPython Software License**.

To comply with the terms of the open-source licensing, the license, copyright notice, disclaimer of warranty, and any required source code of any third-party software are made available with the Data Build Manager.

3001 Summer Street
Stamford CT 06926-0700
USA

www.pitneybowes.com