Clementine® 8.0 User’s Guide
Clementine is the SPSS enterprise-strength data mining workbench. Clementine helps organizations improve customer and citizen relationships through an in-depth understanding of data. Organizations use the insight gained from Clementine to retain profitable customers, identify cross-selling opportunities, attract new customers, detect fraud, reduce risk, and improve government service delivery.

Clementine’s visual interface invites users’ specific business expertise, which leads to more powerful predictive models and shortens time-to-solution. Clementine offers many modeling techniques, such as prediction, classification, segmentation, and association detection algorithms. Once models are created, Clementine Solution Publisher enables their delivery enterprise-wide to decision makers or to a database.

**Compatibility**

Clementine is designed to operate on computer systems running Windows Me, Windows XP, Windows 2000, Windows NT 4.0 with Service Pack 6 or higher.

**Serial Numbers**

Your serial number is your identification number with SPSS Inc. You will need this serial number when you contact SPSS Inc. for information regarding support, payment, or an upgraded system. The serial number was provided with your Clementine system.
Customer Service

If you have any questions concerning your shipment or account, contact your local office, listed on the SPSS Web site at http://www.spss.com/worldwide/. Please have your serial number ready for identification.

Training Seminars

SPSS Inc. provides both public and onsite training seminars. All seminars feature hands-on workshops. Seminars will be offered in major cities on a regular basis. For more information on these seminars, contact your local office, listed on the SPSS Web site at http://www.spss.com/worldwide/.

Technical Support

The services of SPSS Technical Support are available to registered customers. Student Version customers can obtain technical support only for installation and environmental issues. Customers may contact Technical Support for assistance in using Clementine products or for installation help for one of the supported hardware environments. To reach Technical Support, see the SPSS Web site at http://www.spss.com, or contact your local office, listed on the SPSS Web site at http://www.spss.com/worldwide/. Be prepared to identify yourself, your organization, and the serial number of your system.

Tell Us Your Thoughts

Your comments are important. Please let us know about your experiences with SPSS products. We especially like to hear about new and interesting applications using Clementine. Please send e-mail to suggest@spss.com or write to SPSS Inc., Attn.: Director of Product Planning, 233 South Wacker Drive, 11th Floor, Chicago, IL 60606-6412.

Contacting SPSS

If you would like to be on our mailing list, contact one of our offices, listed on our Web site at http://www.spss.com/worldwide/.
# Contents

1 **What's New in Clementine 8.0?** 1  
   Welcome to Clementine 8.0 .................................................. 1  
   New Features ......................................................................... 1  
   Changes Since Clementine 7.0 .............................................. 12  

2 **Clementine Overview** 17  
   Getting Started ....................................................................... 17  
   Clementine at a Glance .......................................................... 24  
   Setting Clementine Options .................................................. 31  
   Automating Clementine .......................................................... 39  

3 **Understanding Data Mining** 41  
   Data Mining Overview ............................................................. 41  
   Machine-Learning Techniques .............................................. 42  
   Assessing Potential Data Mining Applications ..................... 50  
   A Strategy for Data Mining ..................................................... 52  
   Tips ...................................................................................... 55  

4 **Building Streams** 57  
   Stream-Building Overview ..................................................... 57  
   Building Data Streams .......................................................... 58
5  Source Nodes

Overview ................................................. . 97
Variable File Node ........................................ . 98
Fixed File Node ........................................... . 101
Setting Data Storage for Text Fields ......................... . 104
Database Node ............................................. . 106
SPSS Import Node ........................................... . 111
SAS Import Node ............................................ . 113
User Input Node ............................................ . 114
Common Source Node Tabs ................................... . 119

6  Record Operations Nodes

Overview of Record Operations ................................. . 125
Select Node ................................................ . 126
Sample Node ............................................... . 127
Balance Node .............................................. . 129
Aggregate Node ............................................. . 131
Sort Node .................................................. . 134
Merge Node ............................................... . 135
Append Node .............................................. . 143
Distinct Node .............................................. . 145

7  Field Operations Nodes

Field Operations Overview .................................... . 147
Type Node ................................................... . 148
<table>
<thead>
<tr>
<th>Filter Node</th>
<th>164</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derive Node</td>
<td>167</td>
</tr>
<tr>
<td>Filler Node</td>
<td>181</td>
</tr>
<tr>
<td>Reclassify Node</td>
<td>184</td>
</tr>
<tr>
<td>Binning Node</td>
<td>188</td>
</tr>
<tr>
<td>Set to Flag Node</td>
<td>197</td>
</tr>
<tr>
<td>History Node</td>
<td>199</td>
</tr>
<tr>
<td>Field Reorder Node</td>
<td>201</td>
</tr>
</tbody>
</table>

### 8 Building CLEM Expressions

- What Is CLEM? | 205
- Using the Expression Builder | 210
- Types of CLEM Functions | 216
- Common Uses of CLEM. | 217

### 9 Handling Missing Values

- Overview of Missing Values | 223
- Specifying Missing Values | 224
- Treating Missing Values | 225
- CLEM Functions for Missing Values | 228

### 10 Graph Nodes

- Graph Nodes Overview | 231
- Building Graphs | 236
11 Modeling Nodes

Overview of Modeling Nodes .......................................................... 307
Neural Net Node ................................................................. 311
Kohonen Node ................................................................. 324
C5.0 Node ................................................................. 329
Linear Regression Node .......................................................... 334
GRI Node ................................................................. 339
Apriori Node ................................................................. 341
K-Means Node ................................................................. 345
Logistic Regression Node .......................................................... 348
Factor Analysis/PCA Node ........................................................ 359
TwoStep Cluster Node .......................................................... 364
C&R Tree Node ................................................................. 367
Sequence Node ................................................................. 373
12 Generated Models

Overview of Generated Models ........................................ 381
Working with Generated Models in the Generated Models Palette .... 382
Using Generated Models in Streams .................................. 385
Using the Generated Model Browsers .................................. 386
Generated Net Node .................................................... 388
Logistic Regression Equation Node .................................... 392
Linear Regression Equation Node ...................................... 398
Factor Equation Node ................................................... 403
Unrefined Rule Model .................................................... 407
Generated Ruleset Node .................................................. 412
Generated Decision Tree Node ......................................... 418
Generated Cluster Models ............................................... 428
Generated Sequence Rules Node ....................................... 454

13 Exporting Models

Exporting Overview ...................................................... 463
PMML Export ............................................................. 465
C Code Export ............................................................ 465
Exporting to PredictiveMarketing ...................................... 481
Exporting to Cleo ........................................................ 484

14 Output Nodes

Overview of Output Nodes ............................................. 487
Working with Output ..................................................... 488
15 SuperNodes
16 Projects and Reports 575

Introduction to Projects .................................................. 575
Building a Project ............................................................. 579
Building a Report ............................................................ 586

17 Batch Mode Execution 591

Introduction to Batch Mode ............................................. 591
Working in Batch Mode .................................................... 592

18 Scripting in Clementine 597

Introduction to Scripting ............................................... 597
Scripting in the User Interface ........................................ 599
Scripting in Batch Mode .................................................. 606

19 Clementine External Module Interface 607

Introduction to the Clementine External Module Interface .......... 607
How CEMI Works ............................................................ 607
System Architecture ........................................................ 608
Specification File ........................................................... 609
Restrictions ................................................................. 615
Example Specification File .............................................. 616
CEMI Node Management ................................................. 624
Tips for Writing External Programs .................................. 630
20 Application Examples

Overview .......................................................... 631
Condition Monitoring Example .................................. 631
Fraud Detection Example ......................................... 638
Retail Example ...................................................... 647
Market Basket Analysis Example ............................... 652

Appendices

A CLEM Language Reference 663

CLEM Reference Overview ...................................... 663
CLEM Datatypes .................................................... 663
Operator Precedence .............................................. 668
Parameters ......................................................... 669
Functions Reference .............................................. 670
Obsolete Features and Functions ............................... 696

B Scripting Language Reference 697

Scripting Reference Overview .................................... 697
Scripting Syntax .................................................... 697
Setting Properties and Parameters ............................. 700
Parameters in Scripting .......................................... 701
Using CLEM in Scripts. ........................................... 704
Creating Nodes and Streams ..................................... 704
E CEMI Specification File Reference 799

Specification File Overview ........................................ 799
Notation ............................................................................. 800
Node Specification .......................................................... 800
Core Specification ............................................................ 804

Glossary 841

Index 855

xiv
Welcome to Clementine 8.0

Welcome to release 8.0 of the Clementine data mining workbench. This release expands the functionality of the toolkit to include new data preparation tasks, such as field recoding, field reordering, and new join functionality. Back-end improvements include the ability to load data in bulk to a database, the ability to work directly between Clementine Server and SPSS Server, and additional support for SQL generation. The implementation of a graphical view for cluster models provides a counterpart to the Tree Viewer incorporated in release 7.0 for tree models.

Read on for information regarding the specific changes in this release. If you are an old friend of Clementine, please see the section “Changes Since Clementine 7.0” for more information on taking advantage of the enhanced functionality in this release.

New Features

This release includes new features and enhancements designed to expand data transformation, streamline your work, and support your efforts to conduct thorough, insightful data mining.

New Nodes

The following new nodes are included on the Field Ops and Output node palettes:

- **Data Audit node**, for a comprehensive first look at your data.
- **Reclassify node**, used to regroup or collapse categories for numeric or string set fields.
Chapter 1

- **Binning node**, used to automatically recode numeric range fields.
- **Reorder node**, used to customize the natural order of fields downstream.

These new nodes are introduced in the topics that follow.

**New Functionality**

In this release, you can:
- Visualize cluster model results using the Viewer tab for generated Kohonen, K-Means, and TwoStep cluster models.
- Generate encoded passwords for use in scripting and command-line arguments.
- Specify a custom bulk loader program for exporting to a database.
- Generate SQL for decision tree models and rulesets.
- Learn more about the algorithms used in Clementine. See the *Clementine Algorithms Guide* available on the product CD.
- Keep data analysis on the server when transferring between server versions of SPSS and Clementine. For more information, see “Helper Applications” in Chapter 14 on page 550.
- Specify several custom conditions and rules for evaluation charts.
- Perform partial outer joins and anti-joins using new Merge node functionality.
- Filter or rename fields directly from SPSS Procedure, Export, and Publisher nodes. For more information, see “Renaming or Filtering Fields for Export” in Chapter 14 on page 542.

Many of these new features are discussed in the topics that follow.

**New Tools for Data Preparation and Exploration**

This release expands the range of tools available for data preparation and exploration by adding a number of new nodes and enhancements.
**New Data Audit Node**

The Data Audit node provides a comprehensive first look at the data you bring into Clementine. Often used during the initial data exploration, the Data Audit report shows summary statistics as well as histograms and distribution graphs for each data field. The results are displayed in an easy-to-read matrix that can be sorted and used to generate full-size graphs and data preparation nodes.

Figure 1-1  
*Data Audit report in the output browser*

![Data Audit report in the output browser](image)

The Data Audit node is available from the Output nodes palette. For more information, see “Data Audit Node” in Chapter 14 on page 510.

**New Binning Node**

The Binning node enables you to automatically create new set fields based on the values of one or more existing numeric range fields. For example, you can transform a scale income field into a new categorical field containing groups of income as deviations from the mean. In SPSS, this is also known as Automatic Recode. Once you have created bins for the new field, you can generate a Derive node based on the cut points.
The Settings tab provides options for available techniques. The View tab displays cut points established for data previously run through the node.

Figure 1-2
Binning node dialog box: Settings tab

Using the Binning node, you can automatically generate bins (categories) using the following techniques:

- Fixed-width binning
- Tiles (equal counts)
- Mean and standard deviation
- Ranks

The Binning node is available from the Field Ops palette. For more information, see “Binning Node” in Chapter 7 on page 188.
New Reclassify Node

The Reclassify node enables the transformation from one set of discrete values to another. Reclassification is useful for collapsing categories or regrouping data for analysis. For example, you could reclassify the values for Product name into three groups, such as Kitchenware; Clothing, Bath and Linens; and Appliances. You can now reclassify directly from a Distribution node by grouping values and generating a Reclassify node. For more information, see “Using a Distribution Graph” in Chapter 10 on page 261.

The Reclassify node is available from the Field Ops palette. Double-click as usual to specify a variety of options.

Figure 1-3
Reclassify node dialog box
For more information, see “Reclassify Node” in Chapter 7 on page 184.

**New Field Reorder Node**

The Field Reorder node enables you to define the natural order used to display fields downstream. This order affects the display of fields in a variety of places, such as tables, lists, and the field chooser. This operation is useful, for example, when working with wide data sets to make fields of interest more visible.

![Field Reorder Node](image)

Figure 1-4
Reordering to display fields of interest first

The Field Reorder node is available from the Field Ops palette. For more information, see “Field Reorder Node” in Chapter 7 on page 201.
What's New in Clementine 8.0?

**Enhanced Data Merging**

This release includes more sophisticated Merge node capabilities. You can now merge records using the following types of joins:

- Inner join
- Full outer join
- Partial outer join, both left and right joins
- Anti-join, the opposite of an inner join

The Merge node is available on the Record Ops palette. For more information, see “Merge Node” in Chapter 6 on page 135.

**Modeling Enhancements**

Understanding the results of data modeling has never been easier. This release expands support for visualization and exploration of generated models by adding a rich graphical representation of cluster models as well as more flexible evaluation charts.

You will also see even more efficient use of in-database mining in this release with the ability to generate SQL for scoring operations. For more information, see “In-Database Scoring” on page 12.

The product CD also includes published information on the algorithms included in Clementine. You can download the *Clementine Algorithms Guide* from the product CD.

**New Graphical Viewer for Cluster Models**

In this release, you can now view a graphical representation of cluster results on the Viewer tab for the following models:

- Generated Kohonen net node
- Generated K-Means node
- Generated TwoStep Cluster node
Chapter 1

The Cluster Viewer displays summary statistics and distributions for fields between clusters.

Figure 1-5
Sample Cluster Viewer tab with cluster display

For more information, see “Cluster Viewer Tab” in Chapter 12 on page 428.
Note: Some models created before Clementine 8.0 may not display full information on the Viewer tab.

**Lift Calculated for Apriori and GRI**

Lift is now calculated for each rule in an association model. Lift statistics are displayed for each rule as an extra column in the rule browser. Minimum and maximum lift values are calculated and displayed as part of the Analysis section of the model summary. For more information, see “Unrefined Rule Summary Tab” in Chapter 12 on page 409.

**Improved Evaluation Charts**

Evaluation charts now include functionality enabling you to define hit conditions and scoring expressions used in the chart. You can also specify a business rule condition used for display. Lines are now clearly marked in the output, and you can use the mouse to separate the x-axis into bands for generating a variety of nodes.
For more information, see “Evaluation Chart Node” in Chapter 10 on page 290.

**New Deployment Options**

This release expands the number of ways that you can deploy streams and models from Clementine. Two new wizards are included, which streamline the process of bundling stream operations for use in external Web-based applications.

To open the wizards, from the Tools menu choose:
PredictiveMarketing...

or

Cleo...
PredictiveMarketing Wizard. Enables you to create a scenario package containing metadata and operations required by the PredictiveMarketing application. For more information, see “Exporting to PredictiveMarketing” in Chapter 13 on page 481.

Cleo Wizard. Guides you through the process of defining the Web pages of a customized Cleo scenario. It also creates a .jar file containing the required metadata and stream operations. For more information, see “Exporting to Cleo” in Chapter 13 on page 484.

Performance Optimizations

Each release of Clementine includes a number of performance enhancements for in-database mining and server-side analytics. The following topics introduce changes made in this release.
**Bulk Loading**

To increase performance during data export, you can now use a custom bulk loader program specific to your database. Options are available using the Advanced button on Database and Publisher output nodes.

Using this dialog box, you can also fine-tune a number of options, such as row-size or column-wise binding for loading via ODBC and batch size settings for batch commits to the database. For more information, see “Database Output Advanced Options” in Chapter 14 on page 536.

**In-Database Scoring**

Clementine continues to expand its support for in-database mining. In this release, you can now generate SQL from decision trees and rulesets (in addition to linear regression and factor generated models). This allows scoring to be conducted in the database, reducing costly data transfer and calculations on the client.

Specify SQL optimization options in the User Options dialog box. Then specify SQL generation options on the Settings tab for generated ruleset models. For more information, see “Generated Ruleset Settings Tab” in Chapter 12 on page 417. Once you've enabled SQL generation, look for nodes and selected generated models on the canvas to turn purple during execution, indicating the operation is being performed in-database.

For more information on in-database mining and SQL generation, contact Technical Support for a copy of the whitepaper, *SQL Optimization in Clementine*.

**Changes Since Clementine 7.0**

For users who are familiar with Clementine, this release includes several changes that you should note. All changes are covered in the online Help and in the manual, but the most significant are listed here.
What’s New in Clementine 8.0?

CEMI Changes

For those using the Clementine External Module Interface (CEMI) to incorporate their own functionality into Clementine, you can now create and include custom CEMI icons for the node palettes and generated model palettes. For more information, see “Creating CEMI Node Icons” in Chapter 19 on page 627.

Scripting Changes

This release includes several changes to scripting as part of an ongoing effort to expose the full Clementine functionality through the scripting interface in a consistent fashion. Enhancements for this release are discussed in the topics that follow.

Scripting and Batch Mode Changes

Encoded Password Generator

A tool is available through the user interface to generate encoded passwords. Once encoded, you can copy and store the password to script files and command-line arguments. The node property `epassword` used for `databasenode` and `databaseexportnode` stores the encoded password.

To generate an encoded password, from the Tools menu choose:
Encode Password

For more information, see “Generating an Encoded Password” in Appendix D on page 746.

Launch Using a Command File

Command-line launch of Clementine and Clementine Batch has been simplified with the use of the @ argument. To shorten or simplify the `clemb` or `clementine` invocation command, you can use a file that contains one command per line as an alternative to passing all of the commands via the command line. Specify the name of the command file, preceded by the @ symbol. For more information, see “Combining Multiple Arguments” in Appendix C on page 732.
Chapter 1

**Execute Script Selection**

You can now execute selected lines from a stream, SuperNode, or standalone script using a new icon on the toolbar.

**Figure 1-8**
*Toolbar icon used to execute selected lines of a script*

**Changes to Node Properties**

Following are new node properties (also called slot parameters) for this release as well as changes to existing ones.

**New Stream Properties**

- `parameters` enables you to update stream parameters from within a stand-alone script.
- `refresh_source_nodes` is used to refresh Source nodes automatically upon stream execution.

**New Node Properties**

Several new nodes are available in this release. Their properties are available through scripting and are documented in the online Help and manual.

- For the Binning node, new in release 8.0, the complete functionality is available through scripting. For more information, see “Field Operations Nodes” in Appendix D on page 751.
- For the Reclassify node, new in release 8.0, the complete functionality is available through scripting. For more information, see “Field Operations Nodes” in Appendix D on page 751.
- For the Reorder node, new in release 8.0, the complete functionality is available through scripting. For more information, see “Field Operations Nodes” in Appendix D on page 751.
For the Data Audit node, new in release 8.0, the complete functionality is available through scripting. For more information, see “Output Nodes” in Appendix D on page 781.

**New Properties**

- For the Database Export and Publisher nodes, there are several new properties for the ability to bulk load data to a database. For more information, see “Output Nodes” in Appendix D on page 781.
- For the Variable File and Fixed File source nodes, there are two new properties (`invalid_char_mode` and `invalid_char_replacement`) used to remove or replace invalid characters. For more information, see “Source Nodes” in Appendix D on page 742.

**Deprecated Node Properties**

- `full_out_join` for the Merge node has been replaced by `join` where the type of join can be set to one of the following: `Inner`, `FullOuter`, `PartialOuter`, or `Anti`. 

Clementine Overview

Getting Started

As a data mining tool that combines advanced modeling technology with ease of use, Clementine helps you discover and predict interesting and valuable relationships within your data. You can use Clementine for decision-support activities, such as:

- Creating customer profiles and determining customer lifetime value.
- Detecting and predicting fraud in your organization.
- Determining and predicting valuable sequences in Web-site data.
- Predicting future trends in sales and growth.
- Profiling for direct mailing response and credit risk.
- Performing churn prediction, classification, and segmentation.
- Sifting through vast quantities of data from automation and discovering useful patterns.

These are just a sampling of the many ways that you can use Clementine to extract valuable information from your data. Essentially, if you have the data and your data contain the right information, Clementine will help you find answers to your questions.
Chapter 2

Installing Clementine

When you purchased Clementine, you received an installation package containing a combination of the following CD-ROMs, depending on which version of Clementine that you purchased:

- **Clementine Standalone/Client.** This CD installs the Clementine Standalone/Client version.

- **SPSS Data Access Pack.** This CD contains the SPSS Data Access Pack, which installs a set of data access drivers used to access various types of databases.

- **Clementine Server.** This CD installs the Clementine Server version.

- **Clementine Batch (Optional).** This CD installs the batch version for Clementine Server and standalone Clementine Batch.

- **Clementine Solution Publisher Runtime (Optional).** This add-on component installs an environment that allows you to run the Clementine Solution Publisher. Clementine Solution Publisher Runtime will be included only if you have purchased this option.

- **Clementine Application Templates (Optional).** This CD provides vertical market templates, including data, streams, and documentation.

- **Clementine Application Templates for Security (Optional).** This add-on product offers stream templates, documentation, and sample data that give you a head start in selected security applications.

To install Clementine, insert the product CD into your CD-ROM drive. From the AutoPlay menu, choose Install Clementine. The instructions will guide you through the installation process. For more information about installing Clementine Client, Clementine Server, Clementine Batch, SPSS Data Access Pack, and Clementine Solution Publisher Runtime, see the installation documents included on the applicable CD-ROMs.

**System Requirements**

The system requirements for installing the Clementine Client version are:

- **Hardware.** Pentium-compatible processor or higher and a monitor with 1024 x 768 resolution or higher (support for 65,536 colors is recommended). A CD-ROM drive for installation is also required.
Clementine Overview

- **Software.** Installing Clementine installs the Java Virtual Machine: Sun Java Runtime Environment 1.4.1_02. In order to run the online Help system, you should have Internet Explorer version 5.x or Netscape 6.


- **Minimum free disk space.** 320MB.

- **Minimum RAM.** 256MB are recommended. 512MB are recommended when using Clementine Application Templates (CATs) or other large data sets.

**Installation Procedure**

These installation instructions apply to the client version of Clementine for Windows, including Clementine Server or Standalone clients. To install Clementine on Windows NT or Windows 2000, you must be logged in to your computer with administrator privileges.

**To install Clementine:**

- Insert the CD into the CD-ROM drive.
- From the AutoPlay menu that appears, choose Install Clementine 8.0.
- Follow the instructions that appear on the screen.

When installing Clementine, you will be prompted for a license code, which should be included in your Clementine kit. If you cannot find your license code, call your sales representative.

After you have entered your serial number and license code, you will be prompted to select which Clementine add-on components you want to install.
Once you have completed the installation procedure, a new program item—Clementine 8.0—will be added to the Start menu. This item includes Clementine and a demos folder. *Note:* Installing Clementine 8.0 does not automatically overwrite earlier installations of Clementine 7.0 or 7.5. You will need to uninstall these using the Windows Control Panel.

After installing Clementine, you may have additional questions regarding database access and connections to the server.

- Contact your system administrator for information about available servers, user IDs, and passwords. You can also refer to the *Clementine Server Administrator's Guide*, included on the Server CD-ROM.
- For questions about the SPSS Data Access technology, see the *Getting Started with SPSS Data Access Technology* on the SPSS Data Access Pack CD-ROM. Additional DataDirect documentation is included on the CD-ROM.
Uninstalling Clementine

To remove or modify the current installation of Clementine:

- In the Windows Control Panel, select Add/Remove Programs.
- From the list, select Clementine.
- Click Add/Remove.
- The InstallShield Wizard will automatically appear, with options for modifying or removing the installation.

*Note:* For Windows NT or Windows 2000, you must be logged in to your computer with administrator privileges to uninstall program files.

Starting Clementine

Once you have installed Clementine, you can get started by launching the application.

To run Clementine:

- From the Start menu choose:
  Programs
  Clementine
  Clementine

- If you have installed Clementine properly, the main Clementine window will appear after a few seconds.

Launching from the Command Line

Using the command line of your operating system, you can launch the Clementine user interface. From both client and server computers, you can launch Clementine using the following steps:

- Open a DOS window or command prompt window.
Chapter 2

Type the command clementine as well as any arguments (flags) used to load streams, execute scripts, and connect to a server.

*Note:* Clementine can also be launched in batch mode from the command line. For more information, see “Introduction to Batch Mode” in Chapter 17 on page 591.

**Connecting to a Server**

Clementine is a client-server application and can be run against the local computer or a server of your specification. The current connection status is displayed at the bottom left of the Clementine window.

**To connect to a server:**

- Double-click the connection status area of the Clementine window.

  or

- From the Tools menu, select Server Login.

- Using the dialog box, specify options to connect to a server or switch to the local host computer.

**Figure 2-2**

*Server Login dialog box*
**Connection.** Choose Local to launch a local execution server (*clemlocal*). In this mode, the server isn't public and is used only by the current session of Clementine. Choose Network to view a list of servers available on the network and activate the options below.

**Server.** Specify an available server or select one from the drop-down list.

**Port.** Lists the default server port number for the current release. If the default port is not accessible, you should contact your system administrator for an updated port number for the installation of Clementine Server.

**User name.** Enter the user name with which to log in to the server.

**Password.** Enter the password associated with the specified user name.

**Domain.** Specify the domain used to log in to the server.

**Default data path.** Specify a path used for data on the server computer. Click the ellipsis button (...) to browse to the desired location.

**Set as default server.** Select to use the current settings as the default server.

**Changing the Temp Directory**

Some operations performed by Clementine may require temporary files to be created. By default, Clementine uses the system temporary directory to create temp files. You can alter the location of the temporary directory using the following steps.

- Create a new directory called *clem* and subdirectory called *servertemp*.

- Edit *options.cfg*, located in the /config directory of your Clementine installation. Edit the *temp_directory* parameter in this file to read: *temp_directory, "C:/clem/servertemp".*

- After doing this, you must restart the Clementine Server service. You can do this by clicking the Services tab on your Windows Control Panel. Just stop the service and then start it to activate the changes you made. Restarting the machine will also restart the service.

All temp files will now be written to this new directory.
Note: The most common error when attempting to do this is to use the wrong type of slashes. Because of Clementine's UNIX history, we employ forward slashes.

**Clementine at a Glance**

Working in Clementine is working with data. In its simplest form, working with Clementine is a three-step process. First, you read data into Clementine, then run the data through a series of manipulations, and finally send the data to a destination. This sequence of operations is known as a data stream because the data flows record by record from the source through each manipulation and, finally, to the destination—either a model or type of data output. Most of your work in Clementine will involve creating and modifying data streams.

Figure 2-3
A simple stream

At each point in the data mining process, Clementine's visual interface invites your specific business expertise. Modeling algorithms, such as prediction, classification, segmentation, and association detection, ensure powerful and accurate models. Model results can easily be deployed and read into databases, SPSS, and a wide variety of other applications. You can also use the add-on component, Clementine Solution Publisher, to deploy entire data streams that read data into a model and deploy results without a full version of Clementine. This brings important data closer to decision makers who need it.

**Clementine Interface**

The numerous features of Clementine's data mining workbench are integrated by a visual programming interface. You can use this interface to draw diagrams of data operations relevant to your business. Each operation is represented by an icon or node, and the nodes are linked together in a stream representing the flow of data through each operation.
Stream canvas. The stream canvas is the largest area of the Clementine window, and it is where you build and manipulate data streams. You can work with multiple streams at a time in Clementine, either in the same stream canvas or by opening a new stream. Streams are stored in the managers during a session.

Palettes. The palettes are located across the bottom of the Clementine window. Each palette contains a related group of nodes that are available to add to the data stream. For example, the Sources palette contains nodes that you can use to read data into your model, and the Graphs palette contains nodes that you can use to explore your data visually. The Favorites palette contains a default list of nodes frequently used by data miners. As you become more familiar with Clementine, you can customize the contents for your own use.
Managers. At the upper right of the Clementine window are three types of managers. Each tab—Streams, Outputs, and Models—is used to view and manage the corresponding types of objects. You can use the Streams tab to open, rename, save, and delete the streams created in a session. Clementine output, such as graphs and tables, is stored in the Outputs tab. You can save output objects directly from this manager. The Models tab is the most powerful of the manager tabs and contains the results of machine learning and modeling conducted in Clementine. These models can be browsed directly from the Models tab or added to the stream in the canvas.

Projects. The Projects window is located at the lower right of the Clementine window and offers a useful way of organizing your data mining efforts in Clementine. For more information, see “Introduction to Projects” in Chapter 16 on page 575.

Report window. Located below the palettes, the Report window provides feedback on the progress of various operations, such as when data are being read into the data stream.

Status window. Also located below the palettes, the Status window provides information on what the application is currently doing, as well as indications when user feedback is required.

Clementine Toolbars

At the top of the Clementine window, you will find a toolbar of icons that provides a number of useful functions. Following are toolbar buttons and their functions:

- **Create new stream**
- **Open stream**
- **Save stream**
- **Print current stream**
- **Cut node**
- **Copy node**
- **Paste node**
- **Undo last action**
- **Redo**
- **Edit stream properties**
- **Execute current stream**
- **Execute stream selection**
Clementine Overview

Stop stream (Activated only during stream execution)  Add SuperNode
Zoom in (Supernodes only)  Zoom out (Supernodes only)

Customizing the Clementine Window

Using the dividers between various portions of the Clementine interface, you can resize or close tools to meet your preferences. For example, if you are working with a large stream, you can use the small arrows located on each divider to close the palettes, managers window, and projects window. This maximizes the stream canvas, providing enough workspace for large or multiple streams.

Figure 2-5
Maximized stream canvas
As an alternative to closing the nodes palette and manager and project windows, you can use the stream canvas as a scrollable page by moving vertically and horizontally with the blue scrollbars at the side and bottom of the Clementine window.

**Using the Mouse in Clementine**

Some of the operations in the Clementine main window require that your mouse have a third button or wheel. The third button is most often used to click and drag when connecting nodes. If you do not have a three-button mouse, you can simulate this feature by pressing the Alt key while clicking and dragging the mouse.

The most common uses of the mouse in Clementine include the following:

- **Single-click.** Use either the right or left mouse button to select options from menus, open context-sensitive menus, and access various other standard controls and options. Click and hold the button to move and drag nodes.

- **Double-click.** Double-click using the left mouse button to place nodes on the stream canvas and edit existing nodes.

- **Middle-click.** Click the middle mouse button and drag the cursor to connect nodes on the stream canvas. Double-click the middle mouse button to disconnect a node. If you do not have a three-button mouse, you can simulate this feature by pressing the Alt key while clicking and dragging the mouse.

**Using Shortcut Keys**

Many visual programming operations in Clementine have shortcut keys associated with them. For example, you can delete a node by clicking the node and pressing the Delete key on your keyboard. Likewise, you can quickly save a stream by pressing the S key while holding down the Ctrl key. Control commands like this one are indicated by a combination of Ctrl- and another key—for example, Ctrl-S.

There are a number of shortcut keys used in standard Windows operations, such as Ctrl-X to cut. These shortcuts are supported in Clementine along with the following application-specific shortcuts. Select an object in the stream canvas and press the specified keys.
Note: In some cases, old shortcut keys used in Clementine conflict with standard Windows shortcut keys. These old shortcuts are supported with the addition of the Alt key. For example, Alt-Ctrl-C can be used to toggle the cache on and off.

Table 2-1
Supported shortcut keys

<table>
<thead>
<tr>
<th>Shortcut Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl-A</td>
<td>Select all</td>
</tr>
<tr>
<td>Ctrl-X</td>
<td>Cut</td>
</tr>
<tr>
<td>Ctrl-N</td>
<td>New stream</td>
</tr>
<tr>
<td>Ctrl-O</td>
<td>Open stream</td>
</tr>
<tr>
<td>Ctrl-P</td>
<td>Print</td>
</tr>
<tr>
<td>Ctrl-C</td>
<td>Copy</td>
</tr>
<tr>
<td>Ctrl-V</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl-Z</td>
<td>Undo</td>
</tr>
<tr>
<td>Ctrl-Q</td>
<td>Select all nodes downstream of the selected node</td>
</tr>
<tr>
<td>Ctrl-W</td>
<td>Deselect all downstream nodes (toggles with Ctrl-Q)</td>
</tr>
<tr>
<td>Ctrl-E</td>
<td>Execute from selected node</td>
</tr>
<tr>
<td>Ctrl-S</td>
<td>Save current stream</td>
</tr>
<tr>
<td>Alt-Arrow keys</td>
<td>Move selected nodes on the stream canvas in the direction of the arrow used</td>
</tr>
<tr>
<td>Shift-F10</td>
<td>Open the context menu for the selected node</td>
</tr>
</tbody>
</table>

Table 2-2
Supported shortcuts for old hot keys

<table>
<thead>
<tr>
<th>Shortcut Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl-Alt-Z</td>
<td>Zoom out</td>
</tr>
<tr>
<td>Ctrl-Alt-D</td>
<td>Duplicate node</td>
</tr>
<tr>
<td>Ctrl-Alt-L</td>
<td>Load node</td>
</tr>
<tr>
<td>Ctrl-Alt-R</td>
<td>Rename node</td>
</tr>
<tr>
<td>Ctrl-Alt-U</td>
<td>Create User Input node</td>
</tr>
<tr>
<td>Ctrl-Alt-C</td>
<td>Toggle cache on/off</td>
</tr>
<tr>
<td>Ctrl-Alt-F</td>
<td>Flush cache</td>
</tr>
<tr>
<td>Ctrl-Alt-X</td>
<td>Expand SuperNode</td>
</tr>
</tbody>
</table>
**Shortcut Key** | **Function**  
--- | ---  
Ctrl-Alt-Z | Zoom in/zoom out  
Delete | Delete node or connection  
Backspace | Delete node or connection

### Getting Help in Clementine

There are several ways to access the various kinds of help in Clementine:

- **Context-sensitive help.** Click the Help button or icon in most dialog boxes to access a Help topic specifically for the controls in that dialog box.

- **What's This help.** To access general help on nodes and toolbar items, select What's This from the Help menu in Clementine. The cursor changes to a question mark, which you can use to click on any item in the stream canvas or palettes. A Help window will open with information on the selected item.

- **Help on CRISP-DM.** Clementine includes a Help system designed to support the Cross Industry Standard Process for Data Mining. To access this help, select CRISP Help from the Help menu or use the context menu options from the CRISP-DM projects tool to select Help for a particular phase of data mining.

- **Accessibility help.** To view help topics discussing Clementine's accessibility features, select Accessibility Help from the Help menu.

- **Tutorial.** For a “quick start” guide to using Clementine, you can access the online tutorial by selecting Tutorial from the Help menu.

- **Help table of contents.** You can access the entire online Help system by selecting Help Topics from the Help menu. The system includes information on Clementine and data mining as well as all other Help topics.

- **PDF files on the Clementine CDs.** There are numerous PDF files on the product CDs, covering installation, administration, and troubleshooting. Clementine manuals are also included in PDF format on the CDs.

If you cannot find what you are looking for or need additional assistance, please contact SPSS Technical Support at SPSS Technical Support (http://www.spss.com/tech/).
Setting Clementine Options

There are several ways to customize and set options for Clementine:

- Set system options, such as memory usage and locale, by selecting System Options from the Tools menu.
- Set user options, such as fonts, warnings, and optimizations, by selecting User Options from the Tools menu.
- Specify the location of applications that work with Clementine by selecting Helper Applications from the Tools menu.
- Specify the default directories used in Clementine by selecting Set Directory or Set Server Directory from the File menu.

System Options

You can specify the preferred language or locale for Clementine by selecting System Options from the Tools menu. Here you can also set the maximum memory usage for Clementine. Note: Changes made in this dialog box will not take effect until you restart Clementine.

Figure 2-6
System Options dialog box

![System Options dialog box](image)
Maximum memory. Select to impose a limit in megabytes on Clementine’s memory usage. On some platforms, Clementine limits its process size to reduce the toll on computers with limited resources or heavy loads. If you are dealing with large amounts of data, this may cause an “out of memory” error. You can ease memory load by specifying a new threshold.

Use system locale. This option is selected by default and set to English (United States). Deselect to specify another language from the drop-down list of available languages and locales.

Managing Memory

In addition to the Maximum memory setting specified in the System Options dialog box, there are several ways you can optimize memory usage:

- Set up a cache on any nonterminal node so that the data are read from the cache rather than retrieved from the data source when you execute the data stream. This will help decrease the memory load for large data sets. For more information, see “Caching Options for Nodes” in Chapter 4 on page 68.

- Adjust the Maximum set size option in the Stream Properties dialog box. This option specifies a maximum number of members for set fields after which the type of the field becomes typeless. For more information, see “Setting Options for Streams” in Chapter 4 on page 73.

- Force Clementine to free up memory by clicking in the lower right corner of the Clementine window where the memory that Clementine is using and the amount allocated are displayed (xxMB/xxIIMB). Clicking this region turns it a darker shade, after which memory allocation figures will drop. Once the region returns to its regular color, Clementine has freed up all the memory possible.

Setting Default Directories

You can specify the default directory used for file browsers and output by selecting Set Directory or Set Server Directory from the File menu.

- Set Directory. You can use this option to set the working directory. The default working directory is based on the installation path of your version of Clementine, or from the command line path used to launch Clementine. In local mode, the
working directory is the path used for all client-side operations and output files (if they are referenced with relative paths).

- **Set Server Directory.** The Set Server Directory option on the File menu is enabled whenever there is a remote server connection. Use this option to specify the default directory for all server files and data files specified for input or output. The default server directory is $CLEO/data, where $CLEO is the directory in which the Server version of Clementine is installed. Using the command line, you can also override this default by using the -server_directory flag with the clementine command line argument.

### Setting User Options

You can set general options for Clementine by selecting User Options from the Tools menu. These options apply to all streams used in Clementine.

The following types of options can be set by clicking the corresponding tab:

- Display options, such as graph and background colors.
- Notification options, such as model overwriting and error messages.
- Optimization options, such as SQL generation and stream rewriting.

To set stream-specific options, such as decimal separators, time and data formats, and stream scripts, use the Stream Properties dialog box, available from the File and Tools menus.

### Setting Notification Options

Using the Notifications tab of the User Options dialog box, you can set various options regarding the occurrence and type of warnings and confirmation windows in Clementine. You can also specify the behavior of the Outputs and Models tab in the managers window when new output and models are generated.
Warn when a node overwrites a file. Select to warn with an error message when node operations overwrite an existing file.

Warn when a node overwrites a database table. Select to warn with an error message when node operations overwrite an existing database table.

Sound Events. Use the list below to specify whether sounds are used to notify you when an event or error occurs. There are a number of sounds available. Use the ellipsis button (...) to browse for and select a sound. Note: The .wav files used to create sounds in Clementine are stored in the /media/sounds directory of your Clementine installation.

- Mute all sounds. Select to turn off sound notification for all events.
**New Output / New Model.** The options on the right side of this dialog box are used to specify the behavior of the Outputs and Models managers tabs when new items are generated. Select New Output or New Model from the drop-down list to specify the behavior of the corresponding tab. The following options are available:

**Select tab.** Choose whether to switch to the Outputs or Models tab when the corresponding object is generated during stream execution.
- Select Always to switch to the corresponding tab in the managers window.
- Select If generated by current stream to switch only tabs for objects generated by the stream currently visible in the canvas.
- Select Never to restrict the software from switching tabs to notify you of generated output or models.

**Flash tab.** Select whether to flash the Output or Models tab in the managers window when new output or models have been generated.
- Select If not selected to flash the corresponding tab (if not already selected) whenever new objects are generated in the managers window.
- Select Never to restrict the software from flashing tabs to notify you of generated objects.

**Open window (New Output only).** For new output objects, select whether to automatically open an output window upon generation.
- Select Always to always open a new output window.
- Select If generated by current stream to open a new window for output generated by the stream currently visible in the canvas.
- Select Never to restrict the software from automatically opening new windows for generated output.

**Scroll palette to make visible (New Model only).** Select whether to automatically scroll the Models tab in the managers window to make the most recent model visible.
- Select Always to enable scrolling.
- Select If generated by current stream to scroll only for objects generated by the stream currently visible in the canvas.
- Select Never to restrict the software from automatically scrolling the Models tab.

**Replace previous model (New Model only).** Select to overwrite previous iterations of the same model.
Click Default Values to revert to the system default settings for this tab.

**Setting Display Options**

Using the Display tab of the User Options dialog box, you can set options for the display of fonts and colors in Clementine.

*Figure 2-8*  
Setting display options

**Standard Fonts and Colors.** Options in this control box are used to specify the color scheme of Clementine and the size of fonts displayed. Options selected here are not applied until you close and restart the software.

- **Use Clementine defaults.** Select to use the default blue-themed Clementine interface.
- **Use Windows settings.** Select to use the Windows display settings on your computer. This may be useful for increased contrast in the stream canvas and palettes.

- **Small node font size.** Specify a font size to be used in the node palettes and when small nodes are displayed in the stream canvas.

- **Large node font size.** Specify a font size to be used when large (standard) nodes are displayed in the stream canvas.

*Note:* Node size for a stream can be specified on the Layout tab of the Stream Properties dialog box.

**Custom Colors.** For each of the items listed in the table, select a color from the drop-down list. To specify a custom color, scroll to the bottom of the color drop-down list and select Color.

**Chart Category Color Order.** This table lists the currently selected colors used for display in newly created graphs. The order of the colors reflects the order in which they will be used in the chart. For example, if a set field used as a color overlay contains four unique values, then only the first four colors listed here will be used. You can specify different colors using the drop-down list for each color number. To specify a custom color, scroll to the bottom of the drop-down list and select Color. Changes made here do not affect previously created graphs.

Click Default Values to revert to the system default settings for this tab.

### Setting Optimization Options

Using the Optimizations tab of the User Options dialog box, you can optimize Clementine performance during stream execution. Note that Server optimization settings in `options.cfg` override any settings in the Client version.
Enable stream rewriting. Select this option to enable stream rewriting in Clementine. Two types of rewriting optimizations are available, and you can select one or both. Stream rewriting effectively reorders the nodes in a stream behind the scenes for more efficient execution by Clementine Server without altering stream semantics.

- **Optimize SQL generation.** This method of stream rewriting allows execution within the database of key stream operations. When this option is selected, Clementine attempts to reorder nodes in the stream so that more operations can then be “pushed back” using SQL generation for execution in the database. Not only can the database perform operations more efficiently than Clementine, but such “push-backs” are also likely to reduce the size of the data set returned to Clementine for processing. This can reduce network traffic and speed stream operations.

For more information on SQL generation, contact Technical Support for a copy of the technical whitepaper *SQL Optimization in Clementine*.

- **Optimize other execution.** This method of stream rewriting increases the efficiency of operations within Clementine (those that cannot be delegated to the database). Optimization is achieved by reducing the amount of data in the stream as early as possible. While maintaining data integrity, the stream is rewritten to push
operations closer to the data source, thus reducing data downstream for costly operations, such as joins.

**Generate SQL.** Specify to perform some processing of a stream with an ODBC source in the database using SQL code to generate execution processes in sequential order. When Optimize SQL Generation is also selected, the order of stream operations may be shifted behind the scenes (stream rewriting) to optimize operation “push-backs.” These options are unavailable when running Clementine in local mode. When operations for a node have been passed back to the database, the node will be highlighted in purple.

**Show status for records.** Select whether Clementine reports records as they arrive at terminal nodes. Specify a number used for updating the status every $N$ records.

Click Default Values to revert to the system default settings for this tab.

---

**Automating Clementine**

Since advanced data mining can be a complex and sometimes lengthy process, Clementine includes several types of coding and automation support.

- **Clementine Language for Expression Manipulation** (CLEM) is a language for analyzing and manipulating the data that flows along Clementine streams. Data miners use CLEM extensively in stream operations to perform tasks as simple as deriving profit from cost and revenue data or as complex as transforming Web-log data into a set of fields and records with usable information. For more information, see “What Is CLEM?” in Chapter 8 on page 205.

- **Scripting** is a powerful tool for automating tedious processes in the user interface and working with objects in batch mode. Scripts can perform the same kinds of actions that users perform with a mouse or a keyboard. You can set options for nodes and perform derivations using a subset of CLEM. You can also specify output and manipulate generated models. For more information, see “Introduction to Scripting” in Chapter 18 on page 597.

- **Batch mode** enables you to use Clementine in a noninteractive manner by running Clementine with no visible user interface. Using scripts, you can specify stream and node operations as well as modeling parameters and deployment options. For more information, see “Introduction to Batch Mode” in Chapter 17 on page 591.
Chapter 3

Understanding Data Mining

Data Mining Overview

Through a variety of techniques, data mining identifies nuggets of information in bodies of data. Data mining extracts information in such a way that it can be used in areas such as decision support, prediction, forecasts, and estimation. Data is often voluminous but of low value and with little direct usefulness in its raw form. It is the hidden information in the data that has value.

Terms. The terms attribute, field, and variable refer to a single data item common to all cases under consideration. A collection of attribute values that refers to a specific case is called a record, an example, or a case.

Technologies and techniques. In data mining, success comes from combining your (or your expert's) knowledge of the data with advanced, active analysis techniques in which the computer identifies the underlying relationships and features in the data. The process of data mining generates models from historical data that are later used for predictions, pattern detection, and more. The techniques for building these models are called machine learning, or modeling.

Clementine includes a number of machine-learning and modeling technologies, including rule induction, neural networks, association rule discovery, sequence detection, and clustering. It also includes many facilities that let you apply your expertise to the data:

- Data manipulation. Constructs new data items derived from existing ones and breaks down the data into meaningful subsets. Data from a variety of sources can be merged and filtered.

- Browsing and visualization. Displays aspects of the data using the Data Audit node to perform and initial audit including graphs and statistics. For more information, see “Data Audit Node” in Chapter 14 on page 510. Advanced
visualization includes interactive graphics, which can be exported for inclusion in project reports.

- **Statistics.** Confirms suspected relationships between variables in the data. Statistics from SPSS can also be used within Clementine.
- **Hypothesis testing.** Constructs models of how the data behaves and verifies these models.

Typically, you will use these facilities to identify a promising set of attributes in the data. These attributes can then be fed to the modeling techniques, which will attempt to identify underlying rules and relationships.

**Machine-Learning Techniques**

Clementine offers a wide variety of machine-learning techniques. These techniques are summarized below.

**Neural Networks**

Neural networks are simple models of the way the nervous system operates. The basic units are neurons, and they are typically organized into layers, as illustrated in the following figure.

*Figure 3-1 Structure of a neural network*
Input data is presented to the first layer, and values are propagated from each neuron to every neuron in the next layer. The values are modified during transmission by **weights**. Eventually, a result is delivered from the output layer.

Initially, all weights are random, and the answers that come out of the net are probably nonsensical. The network learns through **training**. Examples for which the output is known are repeatedly presented to the network, and the answers it gives are compared to the known outcomes. Information from this comparison is passed back through the network, gradually changing the weights. As training progresses, the network becomes increasingly accurate in replicating the known outcomes. Once trained, the network can be applied to future cases where the outcome is unknown.

**Rule Induction**

One of the problems with neural networks is that the way a trained network makes its decision is opaque. Because the information encoded by the network is simply a collection of numbers, it is very difficult to work out the reasoning that goes into its decision-making process. Neural networks are sometimes referred to as **black boxes** because of this problem.

**Rule induction** is a complimentary technique. Working either from the complete data set or a subset, induction creates a decision tree representing a **rule** for how to classify the data into different outcomes. The tree's structure, and hence the rule's reasoning process, is open and explicit and can be browsed.

**Figure 3-2**
*Simple decision tree*

```
        mot
       /  
      no   yes
     /     /
   DON'T BUY   mileage
  /     
 high     low
       / 
    BUY

   age
  /     
old  recent
     / 
DON'T BUY  BUY
```
Another strength of induction is that the process will automatically include in its rule only the attributes that really matter in making a decision. Attributes that do not contribute to the accuracy of the tree are ignored. This can yield very useful information about the data and can be used in Clementine to reduce the data to only relevant fields before training another learning technique, such as a neural net.

Decision trees such as the one above can be converted into a collection of if-then rules (a ruleset), which in many cases show the information in a more comprehensible form. The decision tree presentation is useful when you want to see how attributes in the data can split or partition the population into subsets relevant to the problem. The ruleset presentation is useful if you want to see how particular groups of items relate to a specific conclusion. For example, the following rule gives us a profile for a group of cars that is worth buying:

IF mot = 'yes'
AND mileage = 'low'
THEN -> 'BUY'.

**Kohonen Networks**

Kohonen networks are a type of neural network that perform clustering. The basic units are neurons, and they are organized into two layers: the input layer and the output layer (also called the output map). All of the input neurons are connected to all of the output neurons, and these connections have strengths, or weights, associated with them.

The output map is a two-dimensional grid of neurons, with no connections between the units. Shown below is a $3 \times 4$ map, although typically maps are larger than this.
Input data is presented to the input layer, and the values are propagated to the output layer. Each output neuron then gives a response. The output neuron with the strongest response is said to be the **winner** and is the answer for that input.

Initially, all weights are random. In order to train, an input pattern is shown and the winner adjusts its weights in such a way that it reacts even more strongly the next time it sees that (or a very similar) record. Also, its **neighbors** (those neurons surrounding it) adjust their weights so that they also react more positively. All of the input records are shown, and weights are updated accordingly. This process is repeated many times until the changes become very small.

When the network is fully trained, records that are similar should appear close together on the output map, whereas records that are vastly different will appear far apart.

### Association Rules

**Association rules** associate a particular conclusion (the purchase of a particular product) with a set of conditions (the purchase of several other products). For example, the rule

\[
\text{beer} \leq \text{cannedveg} \& \text{frozenmeal} \ (173, \ 17.0\%, \ 0.84)
\]
Chapter 3

states that beer often occurs when cannedveg and frozenmeal occur together. The rule is 84% reliable and applies to 17% of the data, or 173 records. Association rule algorithms automatically find the associations that you could find manually using visualization techniques, such as the Web node.

Figure 3-4
Web node showing associations between market basket items

The advantage of association rule algorithms over the more standard decision tree algorithms (C5.0 and C&R Trees) is that associations can exist between any of the attributes. A decision tree algorithm will build rules with only a single conclusion, whereas association algorithms attempt to find many rules, each of which may have a different conclusion.

The disadvantage of association algorithms is that they are trying to find patterns within a potentially very large search space and, hence, can require much more time to run than a decision tree algorithm. The algorithms use a generate and test method for finding rules—simple rules are generated initially, and these are validated against the data set. The good rules are stored and all rules, subject to various constraints, are
then specialized. **Specialization** is the process of adding conditions to a rule. These
new rules are then validated against the data, and the process iteratively stores the
best or most interesting rules found. The user usually supplies some limit to the
possible number of antecedents to allow in a rule, and various techniques based on
information theory or efficient indexing schemes are used to reduce the potentially
large search space.

At the end of the processing, a table of the best rules is presented. Unlike a
decision tree, this set of association rules cannot be used directly to make predictions
in the way that a standard model (such as a decision tree or a neural network) can.
This is due to the many different possible conclusions for the rules. Another level
of transformation is required to transform the association rules into a classification
ruleset. Hence, the association rules produced by association algorithms are known as **unrefined models**. Although the user can browse these unrefined models, they
cannot be used explicitly as classification models unless the user tells the system to
generate a classification model from the unrefined model. This is done from the
browser through a Generate menu option.

Clementine provides three association rule algorithms:

- **GRI** can handle numeric and symbolic inputs but only symbolic outputs. For
  more information, see “GRI Node” in Chapter 11 on page 339.

- **Apriori** can handle only symbolic inputs and symbolic outputs. Apriori can make
  use of clever subsetting techniques to speed up its search because it uses only
  symbolic attributes and can thus be more efficient when used with symbolic data.
  For more information, see “Apriori Node” in Chapter 11 on page 341.

- The **Sequence** node discovers sequential patterns in time-oriented data. A
  **sequence** is a list of item sets that tend to occur in a predictable order. The
  Sequence node detects frequent sequences and creates a generated model node
  that can be used to make predictions. For more information, see “Sequence
  Node” in Chapter 11 on page 373.

**Statistical Models**

Statistical models use mathematical equations to encode information extracted from
the data. Linear regression models attempt to find a straight line or surface through
the range of input fields that minimizes the discrepancies between predicted and
observed output values.
Logistic regression models are somewhat more complicated but use a similar strategy to generate equations for predicting probabilities associated with each possible value of a symbolic output field.

Statistical models have been around for some time and are relatively well understood mathematically. They represent basic models that assume fairly simple kinds of relationships in the data. In some cases, they can give you adequate models very quickly. Even for problems in which more flexible machine-learning techniques (such as neural networks) can ultimately give better results, you can use statistical models as baseline predictive models to judge the performance of advanced techniques.

**Clustering Models**

Clustering models focus on identifying groups of similar records and labeling the records according to the group to which they belong. This is done without the benefit of prior knowledge about the groups and their characteristics. In fact, you may not even know exactly how many groups to look for. This is what distinguishes clustering models from the other machine-learning techniques available in Clementine—there is
no predefined output or target field for the model to predict. These models are often referred to as unsupervised learning models, since there is no external standard by which to judge the model's classification performance. There are no right or wrong answers for these models. Their value is determined by their ability to capture interesting groupings in the data and provide useful descriptions of those groupings.

Clustering methods are based on measuring distances between records and between clusters. Records are assigned to clusters in a way that tends to minimize the distance between records belonging to the same cluster.

Clementine includes three methods for clustering. You have already seen how Kohonen networks can be used for clustering. For more information, see “Kohonen Networks” on page 44. K-Means clustering works by defining a fixed number of clusters and iteratively assigning records to clusters and adjusting the cluster centers. This process of reassignment and cluster center adjustment continues until further refinement can no longer improve the model appreciably. TwoStep clustering works by first compressing the data into a manageable number of small subclusters, then using a statistical clustering method to progressively merge the subclusters into clusters, then merging the clusters into larger clusters, and so on, until the minimum
desired number of clusters is reached. TwoStep clustering has the advantage of automatically estimating the optimal number of clusters for the training data.

Clustering models are often used to create clusters or segments that are then used as inputs in subsequent analyses. A common example of this is the market segments used by marketers to partition their overall market into homogeneous subgroups. Each segment has special characteristics that affect the success of marketing efforts targeted toward it. If you are using data mining to optimize your marketing strategy, you can usually improve your model significantly by identifying the appropriate segments and using that segment information in your predictive models.

Assessing Potential Data Mining Applications

Data mining isn't likely to be fruitful unless the data that you want to use meets certain criteria. The following sections present some of the aspects of the data and application that you should consider.

Is the Data Available?

This may seem like an obvious question, but be aware that although data might be available, it may not be in a form that can be used easily. Clementine can import data from databases (via ODBC) or from files. The data, however, might be held in some other form on a machine that cannot be directly accessed. It will need to be downloaded or dumped in a suitable form before it can be used. It might be scattered among different databases and sources and need to be pulled together. It may not even be online. If it exists only on paper, a data entry phase will be required before you can begin data mining.

Does the Data Cover the Relevant Attributes?

The object of data mining is to identify relevant attributes, so this may seem like an odd question. It is very useful, however, to look at what data is available and try to identify likely relevant factors that are not recorded. In trying to predict ice cream sales, for example, you may have a lot of information on retail outlets or sales histories, but you may not have weather and temperature information that is likely to
play a significant role. Missing attributes don't necessarily mean that data mining will not produce useful results, but they can limit the accuracy of resulting predictions.

A quick way of assessing the situation is to perform a comprehensive audit of your data. Before moving on, consider attaching a Data Audit node to your data source and executing to generate a full report. For more information, see “Data Audit Node” in Chapter 14 on page 510.

**Is the Data Noisy?**

Data often contains errors or may contain subjective, and therefore variable, judgments. These phenomena are collectively referred to as noise. Sometimes, noise in data is normal. There may well be underlying rules, but they may not hold for 100% of the cases.

Typically, the more noise there is in data, the more difficult it is to get accurate results. However, Clementine's machine-learning methods are able to handle noisy data and have been used successfully on data sets containing up to almost 50% noise.

**Is There Enough Data?**

This is a difficult question to answer. In data mining, it is not necessarily the size of a data set that is important. The representativeness of the data set is far more significant, together with its coverage of possible outcomes and combinations of variables.

Typically, the more attributes that are considered, the more records that will be needed to give representative coverage.

If the data is representative and there are general underlying rules, it may well be that a data sample of a few thousand (or even a few hundred) records will give equally good results as a million—and you will get the results more quickly.

**Is Expertise on the Data Available?**

In many cases, you will be working on your own data and will therefore be highly familiar with its content and meaning. However, if you are working on data, say, for another department of your organization or for a client, it is highly desirable that you have access to experts who know the data. They can guide you in the identification of relevant attributes and can help to interpret the results of data mining, distinguishing
the true nuggets of information from “fool's gold,” or artifacts caused by anomalies in the data sets.

**A Strategy for Data Mining**

As with most business endeavors, data mining is much more effective if done in a planned, systematic way. Even with cutting edge data mining tools, such as Clementine, the majority of the work in data mining requires a knowledgeable business analyst to keep the process on track. To guide your planning, answer the following questions:

- What substantive problem do you want to solve?
- What data sources are available, and what parts of the data are relevant to the current problem?
- What kind of preprocessing and data cleaning do you need to do before you start mining the data?
- What data mining technique(s) will you use?
- How will you evaluate the results of the data mining analysis?
- How will you get the most out of the information that you obtained from data mining?

The typical data mining process can become complicated very quickly. There is a lot to keep track of—complex business problems, multiple data sources, varying data quality across data sources, an array of data mining techniques, different ways of measuring data mining success, and so on.

To stay on track, it helps to have an explicitly defined process model for data mining. The process model guides you through the critical issues outlined above and makes sure that the important points are addressed. It serves as a data mining road map so that you won't lose your way as you dig into the complexities of your data.

The data mining process model recommended for use with Clementine is the Cross-Industry Standard Process for Data Mining (CRISP-DM). As you can tell from the name, this model is designed as a general model that can be applied to a wide variety of industries and business problems.
The CRISP-DM Process Model

The general CRISP-DM process model includes six phases that address the main issues in data mining. The six phases fit together in a cyclical process, illustrated in the following figure.

These six phases cover the full data mining process, including how to incorporate data mining into your larger business practices. The six phases include:

- **Business understanding.** This is perhaps the most important phase of data mining. Business understanding includes determining business objectives, assessing the situation, determining data mining goals, and producing a project plan.

- **Data understanding.** Data provides the “raw materials” of data mining. This phase addresses the need to understand what your data resources are and the characteristics of those resources. It includes collecting initial data, describing data, exploring data, and verifying data quality. The Data Audit node available from the Output nodes palette is an indispensable tool for data understanding. For more information, see “Data Audit Node” in Chapter 14 on page 510.

- **Data preparation.** After cataloging your data resources, you will need to prepare your data for mining. Preparations include selecting, cleaning, constructing, integrating, and formatting data.
Chapter 3

- **Modeling.** This is, of course, the flashy part of data mining, where sophisticated analysis methods are used to extract information from the data. This phase involves selecting modeling techniques, generating test designs, and building and assessing models.

- **Evaluation.** Once you have chosen your models, you are ready to evaluate how the data mining results can help you to achieve your business objectives. Elements of this phase include evaluating results, reviewing the data mining process, and determining the next steps.

- **Deployment.** Now that you've invested all of this effort, it's time to reap the benefits. This phase focuses on integrating your new knowledge into your everyday business processes to solve your original business problem. This phase includes plan deployment, monitoring and maintenance, producing a final report, and reviewing the project.

There are some key points in this process model. First, while there is a general tendency for the process to flow through the steps in the order outlined above, there are also a number of places where the phases influence each other in a nonlinear way. For example, data preparation usually precedes modeling. However, decisions made and information gathered during the modeling phase can often lead you to rethink parts of the data preparation phase, which can then present new modeling issues, and so on. The two phases feed back on each other until both phases have been resolved adequately. Similarly, the evaluation phase can lead you to reevaluate your original business understanding, and you may decide that you've been trying to answer the wrong question. At this point, you can revise your business understanding and proceed through the rest of the process again with a better target in mind.

The second key point is embodied by the outer cyclical arrow surrounding the process, indicating the iterative nature of data mining. You will rarely, if ever, simply plan a data mining project, execute it, and then pack up your data and go home. Using data mining to address your customers' demands is an ongoing endeavor. The knowledge gained from one cycle of data mining will almost invariably lead to new questions, new issues, and new opportunities to identify and meet your customers' needs. Those new questions, issues, and opportunities can usually be addressed by mining your data once again. This process of mining and identifying new opportunities should become part of the way you think about your business and a cornerstone of your overall business strategy.
This introduction gives only a brief overview of the CRISP-DM process model. For complete details on using the model, consult any of the following resources:

- Choose Help on CRISP-DM from the Help menu in Clementine to access the CRISP-DM help system.
- The CRISP-DM Guide included with your Clementine materials.
- Data Mining with Confidence, published by SPSS Inc. This guide is available from the SPSS online bookstore.

**Tips**

Following are some tips for dealing with issues that commonly come up during data mining.

**Induction, Neural Net, or Statistical Models?**

If you're not sure which attributes are important, it often makes sense to use induction first to produce a rule. The rule browser will then let you generate a filter that cuts the data down to only the fields that induction found to be important. This can be used to select a good subset of fields before training a net or statistical model. Alternative approaches include training a network and using the Sensitivity Analysis feature to rank the different fields by their relevance to the outcome or using a linear regression model to perform stepwise, forwards, or backwards field selection.

Statistical methods are usually very quick and relatively uncomplicated. Therefore, they can often be used as baseline models, giving you a target to beat with the more time-consuming machine-learning techniques. Typically, though by no means universally true, neural nets will work better on cases with a numeric outcome, while induction will do better on symbolic decisions.

**Is the Data Balanced?**

Suppose you have two outcomes: low or high. Ninety percent of cases are low, and only 10% are high. Neural networks will respond badly to such biased data. They will learn only the low outcomes and tend to ignore the high ones. Their chance of learning to make accurate predictions is greatly increased if there are roughly equal
numbers of each output value. One way of balancing the data in this example would be to use only one-ninth of the low cases and all of the high cases for training.

**Sampling**

When starting to work on large data sets, initially take smaller samples. This will let you get through more simple experiments more quickly. Once you have a feel for how the data behaves, you can test your hypotheses on the entire set.

**Collecting Exceptions**

When testing models, look closely at the cases where they make the wrong decisions (such cases are called exceptions). Applying Clementine's data analysis facilities to these exceptions can give indications of weaknesses in the original training data, which you can then redress, or clues about how to improve the model.
Building Streams

Stream-Building Overview

Data mining using Clementine focuses on the process of running data through a series of nodes, referred to as a stream. This series of nodes represents operations to be performed on the data, while links between the nodes indicate the direction of data flow. Typically, you use a data stream to read data into Clementine, run it through a series of manipulations, and then send it to a destination, such as an SPSS file or the Clementine Solution Publisher.

For example, suppose that you want to open a data source, add a new field, select records based on values in the new field, and then display the results in a table. In this case, your data stream would consist of four nodes:

- A Variable File node, which you set up to read the data from the data source.
- A Derive node, which you use to add the new, calculated field to the data set.
- A Select node, which you use to set up selection criteria to exclude records from the data stream.
- A Table node, which you use to display the results of your manipulations onscreen.
Building Data Streams

Clementine's unique interface lets you mine your data visually by working with diagrams of data streams. At the most basic level, you can build a data stream using the following steps:

- Add nodes to the stream canvas.
- Connect the nodes to form a stream.
- Specify any node or stream options.
- Execute the stream.

Figure 4-1
Completed stream on the stream canvas
This section contains more detailed information on working with nodes to create more complex data streams. It also discusses options and settings for nodes and streams. For step-by-step examples of stream building using the data shipped with Clementine (demos folder of your program installation), see Chapter 20.

**Working with Nodes**

Nodes are used in Clementine to help you explore data. Various nodes in the workspace represent different objects and actions. You connect the nodes to form streams, which, when executed, let you visualize relationships and draw conclusions. Streams are like scripts—you can save them and reuse them with different data files.

**Nodes Palette**

The palette at the bottom of the Clementine window contains all of the possible nodes used in stream building.

**Figure 4-2**

*Record Ops tab on the nodes palette*

Each tab contains a collection of related nodes used for different phases of stream operations, such as:

- **Sources.** Nodes used to bring data into Clementine.
- **Record Ops.** Nodes used for operations on data records, such as selecting, merging, and appending.
- **Field Ops.** Nodes used for operations on data fields, such as filtering, deriving new fields, and determining the data type for given fields.
- **Graphs.** Nodes used to visualize data before and after modeling. Graphs include plots, histograms, web nodes, and evaluation charts.
- **Modeling.** Nodes representing the powerful modeling algorithms available in Clementine, such as neural nets, decision trees, clustering algorithms, and data sequencing.

- **Output.** Nodes used to produce a variety of output for Clementine data, charts, and model results. Output can be viewed within Clementine for many output nodes or sent directly to another application, such as SPSS or Excel.

**Customizing the Favorites Tab**

The Favorites tab on the nodes palette can be customized to accommodate your usage of Clementine. For example, if you frequently analyze time-series data from a database, you might want to be sure that both the Database source node and the Sequence modeling node are available from the Favorites tab. The Palette Manager enables you to easily make these adjustments. To access the Palette Manager:

- From the Tools menu, select Favorites.

**Figure 4-3**
*Selecting nodes to add to the Favorites tab*

![Palette Manager](image)

**Display “Favorites” tab.** Selected by default, this option controls whether a Favorites tab is displayed on the nodes palette.

Using the check boxes in the *Shown?* column, select whether to include each node on the Favorites tab.
Note: The CEMI tab on the Palette Manager contains options for displaying nodes created using the Clementine External Module Interface (CEMI). For more information, see “CEMI Node Management” in Chapter 19 on page 624.

**Adding Nodes to a Stream**

There are three ways to add nodes to a stream from the nodes palette:

- Double-click a node on the palette. *Note:* Double-clicking a node automatically connects it to the current stream. For more information, see “Connecting Nodes in a Stream” on page 61.
- Drag and drop a node from the palette to the stream canvas.
- Click a node on the palette, and then click on the stream canvas.

Once you have added a node to the stream canvas, double-click the node to display its dialog box. The options that are available depend on the type of node that you are adding. For information about specific controls within the dialog box, click Help.

**Removing Nodes**

To remove a node from the data stream, click it and press the Delete key. Or, right-click and select Delete from the context menu.

**Connecting Nodes in a Stream**

Nodes added to the stream canvas do not form a data stream until they have been connected. Connections between the nodes indicate the direction of the data as it flows from one operation to the next. There are a number of ways to connect nodes to form a stream: double-clicking, using the middle mouse button, or manually.

**To add and connect nodes by double-clicking:**

The simplest way to form a stream is to double-click nodes on the palette. This method automatically connects the new node to the selected node on the stream canvas. For example, if the canvas contains a Database node, you can select this node and then double-click the next node from the palette, such as a Derive node. This
action automatically connects the Derive node to the existing Database node. You can repeat this process until you have reached a terminal node, such as a Histogram or Publisher node, at which point any new nodes will be connected to the last non-terminal node upstream.

Figure 4-4
*Stream created by double-clicking nodes from the palettes*

To connect nodes using the middle mouse button:

On the stream canvas, you can click and drag from one node to another using the middle mouse button. (If your mouse does not have a middle button, you can simulate this by pressing the Alt key on your keyboard while clicking with the mouse from one node to another.)

Figure 4-5
*Using the middle mouse button to connect nodes*

To manually connect nodes:

If you do not have a middle mouse button and prefer to manually connect nodes, you can use the context menu for a node to connect it to another node already on the canvas.

- Select a node and right-click to open the context menu.
From the menu, select Connect.

A connection icon will appear both on the start node and the cursor. Click on a second node on the canvas to connect the two nodes.

Figure 4-6
Connecting nodes using the Connect option from the context menu

Figure 4-7
Connected nodes

When connecting nodes, there are several guidelines to follow. You will receive an error message if you attempt to make any of the following types of connections:

- A connection leading to a source node
- A connection leading from a terminal node
- A node having more than its maximum number of input connections
- Connecting two nodes that are already connected
- Circularity (data returns to a node from which it has already flowed)

**Bypassing Nodes in a Stream**

When you bypass a node in the data stream, all of its input and output connections are replaced by connections that lead directly from its input nodes to its output nodes. If the node does not have both input and output connections, then all of its connections are deleted rather than rerouted.

For example, you might have a stream that derives a new field, filters fields, and then explores the results in a histogram and table. If you want to also view the same graph and table for data before fields are filtered, you can add either new Histogram and Table nodes to the stream or you can bypass the Filter node. When you bypass
the Filter node, the connections to the graph and table pass directly from the Derive node. The Filter node is disconnected from the stream.

**Figure 4-8**  
*Bypassing a previously connected Filter node*

*To bypass a node:*

- On the stream canvas, use the middle mouse button to double-click the node that you want to bypass. Alternatively, you can use Alt-double-click.

  *Note:* You can undo this action using the Undo option on the Edit menu or by pressing Ctrl-Z.

**Adding Nodes in Existing Connections**

When you want to add a node between two connected nodes, you can replace the original connection with two new connections—an input connection that leads to the new node and an output connection that leads from it.
With the middle mouse button, click the connection arrow into which you want to insert the node. Alternatively, you can use Alt-click to simulate a middle mouse button. Continue to hold down the mouse button.

Drag the connection to the node that you want it to include and release the mouse button.

Note: You can remove new connections from the node and restore the original by bypassing the node.
**Deleting Connections between Nodes**

You can delete the connection between nodes using two methods:

- Press and hold down the right mouse button on the connection arrow head.

- From the context menu, select Delete Connection.

![Delete Connection](image)

**Figure 4-11**
*Deleting the connection between nodes in a stream*

Or you can delete a connection as follows:

- Select a node and press F3 on your keyboard to delete all connections.

- Select a node, and from the main menus choose:
  
  - **Edit**
    - **Node**
    - **Disconnect**

**Setting Options for Nodes**

Once you have created and connected nodes, there are several options for customizing nodes. Right-click on a node and select one of the menu options.
Select Edit to open the dialog box for the selected node.
Select Connect to manually connect one node to another.
Select Disconnect to delete all links to and from the node.
Select Rename and Annotate to open the Edit dialog box to the Annotations tab.
Select Copy to make a copy of the node with no connections. This can be added to a new or existing stream.
Select Cut or Delete to remove the selected node(s) from the stream canvas. Note: Selecting Cut allows you to paste nodes, while Delete does not.
Select Load Node to open a previously saved node and load its options into the currently selected node. Note: The nodes must be of identical type.
Select Save Node to save the node’s details in a file. You can load node details only into another node of the same type.
Select Cache to expand the menu, with options for caching the selected node.
Select Data Mapping to expand the menu, with options for mapping data to a new source or specifying mandatory fields.

Select Create SuperNode to expand the menu, with options for creating a SuperNode in the current stream. For more information, see “Creating SuperNodes” in Chapter 15 on page 557.

Select Generate User Input Node to replace the selected node. Examples generated by this node will have the same fields as the current node. For more information, see “User Input Node” in Chapter 5 on page 114.

Select Execute From Here to execute all terminal nodes downstream from the selected node.

**Caching Options for Nodes**

To optimize stream execution, you can set up a cache on any non-terminal node. When you set up a cache on a node, the cache is filled with the data that pass through the node the next time you execute the data stream. From then on, the data are read from the cache rather than the data source.

For example, suppose you have a source node set to read sales data from a database and an Aggregate node that summarizes sales by location. You can set up a cache on the Aggregate node rather than the source node because you want the cache to store the aggregated data rather than the entire data set.

Nodes with caching enabled are displayed with a small document icon at the top right corner. When the data are cached at the node, the document icon is green.
To enable a cache:

- On the stream canvas, right-click the node and choose Cache from the context menu.
- From the caching submenu, choose Enable.
- You can turn the cache off by right-clicking the node and choosing Disable from the caching submenu.

To flush a cache:

A white document icon on a node indicates that its cache is empty. When the cache is full, the document icon becomes solid green. If you want to replace the contents of the cache, you must first flush the cache and then reexecute the data stream to refill it.

- On the stream canvas, right-click the node and choose Cache from the context menu.
- From the caching submenu, choose Flush.
To save a cache:

You can save the contents of a cache as a SPSS data file (*.sav). You can then either reload the file as a cache, or you can set up a node that uses the cache file as its data source. You can also load a cache that you saved from another project.

- On the stream canvas, right-click the node and choose Cache from the context menu.
- From the caching submenu, choose Save Cache.
- In the Save Cache dialog box, browse to the location where you want to save the cache file.
- Enter a name in the File Name text box.
- Be sure that *.sav is selected in the Files of Type drop-down list, and click Save.

To load a cache:

If you have saved a cache file before removing it from the node, you can reload it.

- On the stream canvas, right-click the node and choose Cache from the context menu.
- From the caching submenu, choose Load Cache.
- In the Load Cache dialog box, browse to the location of the cache file, select it, and click Load.

Annotating Nodes

All nodes in Clementine can be annotated in a number of ways. You can annotate a node to provide additional description about that node. For example, you may want to include annotations that provide more information about fields within a node or describe a node's role in a stream. You may also want to add tooltip text for lengthy streams with a number of similar graphs or Derive nodes, for example. This will help you distinguish between nodes on the stream canvas.
**To annotate a node:**

Editing a node opens a tabbed dialog box containing an Annotations tab used to set a variety of annotation options. You can also open the annotations tab directly:

- Right-click on the node on the stream canvas.
- Select Rename and Annotate. The Edit dialog box opens with the Annotations tab visible.

**Figure 4-14**  
Annotations tab options

**Name.** Select Custom to adjust the auto-generated name or create a unique name for the node as displayed on the stream canvas. Auto is selected by default.

**Tooltip text.** Enter text used as a tooltip for nodes on the stream canvas. This is useful when working with a large number of similar nodes.

The main text window can be used to enter lengthy annotations regarding the operations of the node or decisions made in the node. For example, when you are sharing and reusing streams, it is helpful to take notes on decisions such as discarding a field with numerous blanks using a Filter node. Annotating the node stores this
information with the node. You can also choose to include these annotations in a project report created with the projects tool.

**Working with Streams**

Once you have connected source, process, and terminal nodes on the stream canvas, you have created a stream. As a collection of nodes, streams can be saved, annotated, and added to projects. You can also set numerous options for streams, such as optimization, date/time settings, parameters, and scripts. These properties are discussed in the topics that follow.

In Clementine, you can use and modify more than one data stream at a time. The right side of the Clementine window contains the managers tool, which helps you to navigate the streams currently open. To view the managers tool, select **Managers** from the View menu. Then click the Streams tab.

**Figure 4-15**
*Streams tab in the managers tool with context menu options*

From this tab, you can:
- Access streams
- Save streams
- Save streams to the current project
- Close streams
- Open new streams
Right-click on a stream on the Streams tab to access these options.

**Setting Options for Streams**

For the current stream, you can specify a number of options, many of which apply to CLEM expressions.

**To set stream options:**

- From the File menu, select Stream Properties. Alternatively, you can use the context menu on the Streams tab in the managers tool.

- Click the Options tab.
**Calculations in.** Select Radians or Degrees as the unit of measurement to be used in trigonometric CLEM expressions.

**Import date/time as.** Select whether to use date/time storage for date/time fields or whether to import them as string variables.

**Date format.** Select a date format to be used for date storage fields or when strings are interpreted as dates by CLEM date functions.

**Time format.** Select a time format to be used for time storage fields or when strings are interpreted as times by CLEM time functions.
**Display decimal places.** Set a number of decimal places to be used for displaying and printing real numbers in Clementine.

**Decimal symbol.** From the drop-down list, select either a comma (,) or period (.) as a decimal separator.

**Rollover days/mins.** Select whether negative time differences should be interpreted as referring to the previous day or hour.

**Date baseline (1st Jan).** Select the baseline years (always 1st January) to be used by CLEM date functions that work with a single date.

**2-digit dates start from.** Specify the cutoff year to add century digits for years denoted with only two digits. For example, specifying 1930 as the cutoff year will roll over 05/11/02 to the year 2002. The same setting will use the 19th century for dates after 30, such as 05/11/73.

**Maximum set size.** Select to specify a maximum number of members for set fields after which the type of the field becomes typeless. This option is disabled by default, but it is useful when working with large set fields. *Note:* The direction of fields set to typeless is automatically set to none. This means the fields are not available for modeling.

**Limit set size for Neural, Kohonen, and K-Means modeling.** Select to specify a maximum number of members for set fields used in Neural nets, Kohonen nets, and K-Means modeling. The default set size is 20, after which the field is ignored and a warning is raised, providing information on the field in question.

**Ruleset evaluation.** Determine how rulesets are evaluated. By default, rulesets use Voting to combine predictions from individual rules and determine the final prediction. To ensure that rulesets use the first hit rule by default, select First Hit. For more information, see “Generated Ruleset Node” in Chapter 12 on page 412.

**Refresh source nodes on execution.** Select to automatically refresh all source nodes when executing the current stream. This action is analogous to clicking the Refresh button on a source node, except that this option automatically refreshes all source nodes (except User Input nodes) automatically for the current stream.

*Note:* Selecting this option flushes the caches of downstream nodes even if the data hasn’t changed. Flushing occurs only once per execution, though, which means that you can still use downstream caches as temporary storage for a single execution. For example, say that you’ve set a cache midstream after a complex derive operation and...
that you have several graphs and reports attached downstream of this Derive node. When executing, the cache at the Derive node will be flushed and refilled but only for the first graph or report. Subsequent terminal nodes will read data from the Derive node cache.

The options specified above apply only to the current stream. To set these options as the default for all streams in Clementine, click Save As Default.

**Setting Options for Stream Layout**

Using the Layout tab in the stream properties dialog box, you can specify a number of options regarding the display and usage of the stream canvas.

**To set layout options:**

1. From the File menu, choose Stream Properties. Alternatively, from the Tools menu, choose:
   - Stream Properties
   - Layout
2. Click the Layout tab in the stream properties dialog box.
Stream canvas width. Specify the width of the stream canvas in pixels.

Stream canvas height. Specify the height of the stream canvas in pixels.

Stream scroll rate. Specify the scrolling rate for the stream canvas. Higher numbers specify a faster scroll rate.

Icon name maximum. Specify a limit in characters for the names of nodes on the stream canvas.

Icon size. Select whether to display large or small node icons on the stream canvas.

Grid cell size. Select a grid cell size from the drop-down list. This number is used to for aligning nodes on the stream canvas using an invisible grid. The default grid cell size is 0.25.

Snap to Grid. Select to align icons to an invisible grid pattern (selected by default).
The options specified above apply only to the current stream. To set these options as the default for all streams in Clementine, click Save As Default.

**Viewing Stream Execution Messages**

Messages regarding stream operations such as execution, time elapsed for model building, and optimization can be easily viewed using the Messages tab in the stream properties dialog box. Error messages are also reported in this table.

**To view stream messages:**

- From the File menu, choose Stream Properties. Alternatively, from the Tools menu, choose:
  
  Stream
  
  Messages

- Click the Messages tab in the stream properties dialog box.
In addition to messages regarding stream operations, error messages are reported here. When stream execution is terminated due to an error, this dialog box will open to the Messages tab with the error message visible. Additionally, the node with errors is highlighted in red on the stream canvas.
You can save messages reported here for a stream by selecting Save Messages from the save button drop-down list on the Messages tab. You can also clear all messages for a given stream by selecting Clear All Messages from the save button drop-down list.

**Viewing and Setting Stream Parameters**

For each stream in Clementine, you have the ability to set user-defined variables, such as *Minvalue*, whose values can be specified when used in scripting or CLEM expressions. These variables are called **parameters**. You can set parameters for streams, sessions, and SuperNodes. Stream parameters are saved and loaded with the stream diagrams.
**To view and set stream parameters:**

- From the File menu, choose Stream Properties. Alternatively, from the Tools menu, choose:
  
  Stream Properties
  Parameters

- Click the Parameters tab in the stream properties dialog box.

**Figure 4-20**

*Setting parameters for streams*

<table>
<thead>
<tr>
<th>Name</th>
<th>Long name</th>
<th>Storage</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinValue</td>
<td>Minimum value</td>
<td>Integer</td>
<td>50</td>
<td>(no values)</td>
</tr>
<tr>
<td>MaxValue</td>
<td>Maximum value</td>
<td>Integer</td>
<td>75</td>
<td>(no values)</td>
</tr>
<tr>
<td>ServiceCode</td>
<td>Service Code</td>
<td>String</td>
<td>008</td>
<td>Set</td>
</tr>
</tbody>
</table>

**Name.** Parameter names are listed here. You can create a new parameter by entering a name in this field. This name is used for display in the Expression Builder.

**Long name.** Lists the descriptive name for each parameter created.
**Storage.** Select a storage type from the drop-down list. Storage indicates how the data values are stored in the parameter. For example, when working with values containing leading zeros that you want to preserve (such as 008), you should select String as the storage type. Otherwise, the zeros will be stripped from the value. Available storage types are String, Integer, Real, Time, Date, and Timestamp.

**Value.** Lists the current value for each parameter. Adjust the parameter as desired.

**Type (optional).** If you plan to deploy the stream to an external application, select a usage type from the drop-down list. Otherwise, it is advisable to leave the type column as is.

Click the arrows at the right to move the selected parameter further up or down the list of available parameters. Use the delete button (marked with an X) to remove the selected parameter.

These parameters can then be used in CLEM expressions and scripting for any nodes in the stream. They will appear on the Parameters drop-down list in the Expression Builder.

---

**Setting Session Parameters**

Parameters in Clementine can be set for a specific level, such as stream or SuperNode parameters, or they can be specified more globally using session parameters. Parameters set for a session are available to all streams used in a single instance of Clementine (all streams listed on the Streams tab in the managers window). Setting a parameter is like creating a variable, $x$, that can be used in CLEM expressions and scripting. You can supply the name of the variable as well as the value using the dialog boxes provided in Clementine.

**To set session parameters:**

- From the Tools menu, choose Set Session Parameters.
- Use the dialog box that opens in the same manner as the Parameters tab for streams.
Annotating and Renaming Streams

Using the Annotations tab in the stream properties dialog box, you can add descriptive annotations for a stream and create a custom name for the stream. These options are useful especially when generating reports for streams added to the projects tool.

Figure 4-21
Annotating streams

To rename and annotate streams:

- From the File menu, choose Stream Properties. Alternatively, you can right-click a stream in the managers window and select Stream Properties from the menu, or from the Edit menu, select Stream, and then Rename and Annotate.

- Click the Annotations tab in the stream properties dialog box.
Select whether to use the auto-generated stream name, such as Stream1, Stream2, etc., or create a custom name using the text box.

In the main text window, enter any descriptions and click OK or Apply.

**Viewing Global Values for Streams**

Using the Globals tab in the stream properties dialog box, you can view the global values set for the current stream. Global values are created using a Set Globals node to determine statistics such as mean, sum, or standard deviation for selected fields. Once the Set Globals node is executed, these values are then available for a variety of uses in stream operations.

**To set global values for a stream:**

From the File menu, choose Stream Properties. Alternatively, from the Tools menu, choose:
Stream Properties
Globals

Click the Globals tab in the stream properties dialog box.
Building Streams

Figure 4-22
Viewing global values available for the stream

<table>
<thead>
<tr>
<th>Field</th>
<th>MEAN</th>
<th>SLIM</th>
<th>MIN</th>
<th>MAX</th>
<th>SDEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>42.597</td>
<td>12.809</td>
<td>18</td>
<td>67</td>
<td>14.257</td>
</tr>
<tr>
<td>income</td>
<td>27,797.832</td>
<td>9,339,209.71</td>
<td>6,294.21</td>
<td>81,554.6</td>
<td>12,720.541</td>
</tr>
</tbody>
</table>

**Globals available.** Available globals are listed in this table. You cannot edit global values here, but you can clear all global values for a stream using the clear all values button to the right of the table.

**Executing Streams**

Once you have specified the desired options for streams and connected the desired nodes, you can execute the stream by running the data through nodes in the stream. There are several ways to execute a stream within Clementine:

- You can select Execute from the Tools menu.
- You can also execute your data streams by clicking one of the execute buttons on the toolbar. These buttons allow you to execute the entire stream or simply the selected terminal node. For more information, see “Clementine Toolbars” in Chapter 2 on page 26.
You can execute a single data stream by right-clicking a terminal node and choosing Execute from the context menu.

You can execute part of a data stream by right-clicking any non-terminal node and choosing Execute From Here from the context menu, which executes all operations after the selected node.

To halt the execution of a stream in progress, you can click the red stop button on the toolbar or select Stop Execution from the Tools menu.

**Saving Data Streams**

After you have created a stream, you can save it for future reuse.

*To save a stream:*

- From the File menu, choose Save Stream or Save Stream As.
- In the Save dialog box, browse to the folder in which you want to save the stream file.
- Enter a name for the stream in the File Name text box.
- Select Add to project if you would like to add the saved stream to the current project.

Clicking Save stores the stream with the extension *.str in the specified directory.

**Saving States**

In addition to streams, you can save states, which include the currently displayed stream diagram and any generated models that you have created (listed on the Models tab in the managers window).

*To save a state:*

- From the File menu, choose State or Save State.
In the Save dialog box, browse to the folder in which you want to save the state file. Clicking Save stores the state with the extension *.cst in the specified directory.

**Saving Nodes**

You can also save an individual node by right-clicking the node in the stream canvas and choosing Save Node from the context menu. Use the file extension *.nod.

**Saving Multiple Stream Objects**

When you exit Clementine with multiple unsaved objects, such as streams, projects, or the generated models palette, you will be prompted to save before completely closing the software. If you choose to save items, a dialog box will appear with options for saving each object.

*Figure 4-23*

*Saving multiple objects*

- Simply select the check boxes for the objects that you want to save.
- Click OK to save each object in the desired location.
You will then be prompted with a standard Save dialog box for each object. After you have finished saving, the application will close as originally instructed.

**Loading Files**

You can reload a number of saved objects in Clementine.

- Streams (.str)
- States (.cst)
- Models (.gm)
- Models palette (.gen)
- Nodes (.nod)
- Output (.cou)
- Projects (.cpj)

**Opening New Files**

Streams can be loaded directly from the File menu:

- From the File menu, choose Open Stream.

All other file types can be opened using the submenu items available from the File menu. For example, to load a model, from the File menu, choose:

Models
  Open Model or Load Models Palette

When loading streams created with earlier versions of Clementine, some nodes may be out of date. In some cases, the nodes will be automatically updated, and in others you will need to convert them using a utility.

- The Cache File node has been replaced by the SPSS Import node. Any streams that you load containing Cache File nodes will be replaced by SPSS Import nodes.
- The Build Rule node has been replaced by the C&R Tree node. Any streams that you load containing Build Rule nodes will be replaced by C&R Tree nodes.
**Opening Recently Used Files**

For quick loading of recently used files, you can use the options at the bottom of the File menu.

**Figure 4-24**
*Opening recently used options from the File menu*

<table>
<thead>
<tr>
<th>Recent Streams</th>
<th>mailshct3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recent Projects</td>
<td>goodplot</td>
</tr>
<tr>
<td>Recent States</td>
<td>mailshct4_graphschapter cluster</td>
</tr>
<tr>
<td></td>
<td>drugcluster</td>
</tr>
<tr>
<td></td>
<td>drugcluster</td>
</tr>
<tr>
<td></td>
<td>conoleam</td>
</tr>
<tr>
<td></td>
<td>mailshct6</td>
</tr>
</tbody>
</table>

Select Recent Streams, Recent Projects, or Recent States to expand a list of recently used files.

**Mapping Data Streams**

Using the mapping tool, you can connect a new data source to a preexisting stream or template. The mapping tool will not only set up the connection but it will also help you to specify how fields in the new source will replace those in the existing template. Instead of re-creating an entire data stream for a new data source, you can simply connect to an existing stream.

The data mapping tool allows you to join together two stream fragments and be sure that all of the (essential) field names match up properly. A common use is to replace a source node defined in a Clementine Application Template (CAT) with a source node that defines your own data set. In essence, mapping data results simply in the creation of a new Filter node, which matches up the appropriate fields by renaming them.

There are two equivalent ways to map data:

**Select Replacement Node.** This method starts with the node to be replaced. First, you select the node to replace; then, using the Replacement option from the context menu, select the node with which to replace it. This way is particularly suitable for mapping data to a template.
**Map to.** This method starts with the node to be introduced to the stream. First, select the node to introduce; then, using the Map option from the context menu, select the node to which it should join. This way is particularly useful for mapping to a terminal node. *Note:* You cannot map to Merge or Append nodes. Instead, you should simply connect the stream to the Merge node in the normal manner.

In contrast to earlier versions of Clementine, data mapping is now tightly integrated into stream building, and if you try to connect to a node that already has a connection, you will be offered the option of replacing the connection or mapping to that node.

**Mapping Data to a Template**

To replace the data source for a template stream with a new source node bringing your own data into Clementine, you should use the Select Replacement Node option from the Data Mapping context menu option. This option is available for all nodes except Merge, Aggregate, and all terminal nodes. Using the data mapping tool to perform this action helps ensure that fields are matched properly between the existing stream operations and the new data source. The following steps provide an overview of the data mapping process.
Step 1: Specify Essential Fields in the original source node. In order for stream operations to execute properly, essential fields should be specified. In most cases, this step is completed by the template author. For more information, see “Specifying Essential Fields” on page 93.

Step 2: Add new data source to the stream canvas. Using one of Clementine's source nodes, bring in the new replacement data.

Step 3: Replace the template source node. Using the Data Mapping options on the context menu for the template source node, choose Select Replacement Node. Then select the source node for the replacement data.

Figure 4-26
Selecting a replacement source node

Step 4: Check mapped fields. In the dialog box that opens, check that the software is mapping fields properly from the replacement data source to the stream. Any unmapped essential fields are displayed in red. These fields are used in stream operations and must be replaced with a similar field in the new data source in order for downstream operations to function properly. For more information, see “Examining Mapped Fields” on page 94.

After using the dialog box to ensure that all essential fields are properly mapped, the old data source is disconnected and the new data source is connected to the template stream using a Filter node called Map. This Filter node directs the actual mapping of fields in the stream. An Unmap Filter node is also included on the stream canvas. The Unmap Filter node can be used to reverse field name mapping by adding it to the stream. It will undo the mapped fields, but note that you will have to edit any downstream terminal nodes to reselect the fields and overlays.
**Figure 4-27**
*New data source successfully mapped to the template stream*

**Mapping between Streams**

Similar to connecting nodes, this method of data mapping does not require you to set essential fields beforehand. With this method, you simply connect from one stream to another using the data mapping context menu option, **Map to**. This type of data mapping is useful for mapping to terminal nodes and copying and pasting between streams. *Note:* Using the **Map to** option, you cannot map to **Merge**, **Append**, and all types of source nodes.

**Figure 4-28**
*Mapping a stream from its Sort node to the Type node of another stream*
To map data between streams:

- Right-click the node that you want to use for connecting to the new stream.

- From the context menu, select:
  - Data mapping
  - Map to

- Use the cursor to select a destination node on the target stream.

- In the dialog box that opens, ensure that fields are properly matched and click OK.

Specifying Essential Fields

When mapping to a template, essential fields will typically be specified by the template author. These essential fields indicate whether a particular field is used in downstream operations. For example, the existing stream may build a model that uses a field called Churn. In this stream, Churn is an essential field because you could not build the model without it. Likewise, fields used in manipulation nodes, such as a Derive node, are necessary to derive the new field. Explicitly setting such fields as essential helps to ensure that the proper fields in the new source node are mapped to them. If mandatory fields are not mapped, you will receive an error message. If you decide that certain manipulations or output nodes are unnecessary, you can delete the nodes from the stream and remove the appropriate fields from the Essential Fields list.

Note: In general, template streams in the Solutions Template Library already have essential fields specified.

To set essential fields:

- Right-click on the source node of the template stream that will be replaced.

- From the context menu, select Specify Essential Fields.
Using the Field Chooser, you can add or remove fields from the list. To open the Field Chooser, click the icon to the right of the fields list.

**Examining Mapped Fields**

Once you have selected the point at which one data stream or data source will be mapped to another, a dialog box opens for you to select fields for mapping or to ensure that the system default mapping is correct. If essential fields have been set for the stream or data source and they are unmatched, these fields are displayed in red. Any unmapped fields from the data source will pass through the Filter node unaltered, but note that you can map non-essential fields as well.
Original. Lists all fields in the template or existing stream—all of the fields that are present further downstream. Fields from the new data source will be mapped to these fields.

Mapped. Lists the fields selected for mapping to template fields. These are the fields whose names may have to change to match the original fields used in stream operations. Click in the table cell for a field to activate a drop-down list of available fields.

If you are unsure of which fields to map, it may be useful to examine the source data closely before mapping. For example, you can use the Types tab in the source node to review a summary of the source data.
Source Nodes

Overview

Clementine offers simple and powerful methods to gain access to a wide variety of data sources. The Sources palette contains nodes that you can use to import the contents of various flat files as well as connect to the data within ODBC-compliant relational databases. You can also generate synthetic data using the User Input node.

Figure 5-1
Sources palette

The Sources palette contains the following nodes:

- Database—Used to import data using ODBC.
- Variable File—Used for freefield ASCII data.
- Fixed File—Used for fixed-field ASCII data.
- SPSS File—Used to import SPSS files.
- SAS File—Used to import files in SAS format.
- User Input—Used to replace existing source nodes. This node is also available by right-clicking on an existing node.

To start building a stream, add a source node to the stream canvas. Double-click the node to open a tabbed dialog box where you can read in data, view the fields and values, and set a variety of options, such as filter, data types, field direction, and missing value checking. Use the tabs to switch between operations.
On the File tab, each type of source node has unique options for accessing data. These options are discussed in the topics below. Additional tabs, such as Data, Filter, Type, and Annotations, are common to all source nodes and are discussed toward the end of this chapter. For more information, see “Common Source Node Tabs” on page 119.

**Variable File Node**

You can use Variable File nodes to read data from freefield text files (files whose records contain a constant number of fields but a varied number of characters). This type of node is also useful for files with fixed-length header text and certain types of annotations.

During the execution of a stream, the Variable File node first tries to read the file. If the file does not exist or you do not have permission to read it, an error will occur and the execution will end. If there are no problems opening the file, records will be read one at a time and passed through the stream until the entire file is read.
Figure 5-2
Variable File node dialog box

Setting Options for the Variable File Node

**File.** Specify the name of the file. You can enter a filename or click the ellipsis button (...) to select a file. The file path is shown once you have selected a file and its contents are displayed with delimiters in the panel below.

The sample text displayed from your data source can be copied and pasted into the following controls: EOL comment characters and user-specified delimiters. Use Ctrl-C and Ctrl-V to copy and paste.
Chapter 5

Read field names from file. Selected by default, this option treats the first row in the data file as labels for the column. If your first row is not a header, deselect to automatically give each field a generic name, such as Field1, Field2, for the number of fields in the data set.

Specify number of fields. Specify the number of fields in each record. Clementine can detect the number of fields automatically as long as the records are new-line terminated. You can also set a number manually.

Skip header characters. Specify how many characters you want to ignore at the beginning of the first record.

EOL comment characters. Specify characters, such as # or !, to indicate annotations in the data. Wherever one of these characters appears in the data file, everything up to but not including the next new-line character will be ignored.

Strip lead and trail spaces. Select options for discarding leading and trailing spaces in strings on import.

Invalid characters. Select Discard to remove invalid characters from the data input. Select Replace with to replace invalid characters with the specified symbol (one character only). Invalid characters are null (0) characters or any character that does not exist in the server's encoding.

Delimiters. Using the check boxes listed for this control, you can specify which characters, such as the comma (,), define field boundaries in the file. You can also specify more than one delimiter, such as “,” |” for records that use multiple delimiters. The default delimiter is the comma.

Select Allow multiple blank delimiters to treat multiple adjacent blank delimiter characters as a single delimiter. For example, if one data value is followed by four spaces and then another data value, this group would be treated as two fields rather than five.

Quotes. Using the drop-down lists for this control, you can specify how single and double quotation marks are treated on import. You can choose to Discard all quotation marks, Include as text by including them in the field value, or Pair and discard to match pairs of quotation marks and remove them. If a quotation mark is unmatched, you will receive an error message. Both Discard and Pair and discard store the field value (without quotation marks) as a string.
Decimal symbol. Select the type of decimal separator used in your data source. The Stream default is the character selected from the Options tab of the stream properties dialog box. Otherwise, select either Period(,) or Comma(,) to read all data in this dialog box using the chosen character as the decimal separator.

Lines to scan for type. Specify how many lines to scan for specified data types.

At any point when working in this dialog box, click Refresh to reload fields from the data source. This is useful when altering data connections to the source node or when working between tabs on the dialog box.

Fixed File Node

You can use Fixed File nodes to import data from fixed-field text files (files whose fields are not delimited but start at the same position and are of a fixed length). Machine-generated or legacy data is frequently stored in fixed-field format. Using the File tab of the Fixed File node, you can easily specify the position and length of columns in your data.

Setting Options for the Fixed File Node

The File tab of the Fixed File node allows you to bring data into Clementine and specify the position of columns and length of records. Using the data preview pane in the center of the dialog box, you can click to add arrows specifying the breakpoints between fields.
Chapter 5

Figure 5-3
Specifying columns in fixed-field data

File. Specify the name of the file. You can enter a filename or click the ellipsis button (...) to select a file. The file path is shown once you have selected a file and its contents are displayed with delimiters in the panel below.

The data preview pane can be used to specify column position and length. The ruler at the top of the preview window helps measure the length of variables and specify the breakpoint between them. You can specify breakpoint lines by clicking in the ruler area above the fields. Breakpoints can be moved by dragging and discarded by dragging them outside of the data preview region.
- Each breakpoint line automatically adds a new field to the fields table below.
- Start positions indicated by the arrows are automatically added to the Start column in the table below.

**Line oriented.** Select if you want to skip the new-line character at the end of each record.

**Skip header lines.** Specify how many lines you want to ignore at the beginning of the first record. This is useful for ignoring column headers.

**Record length.** Specify the number of characters in each record.

**Decimal symbol.** Select the type of decimal separator used in your data source. The Stream default is the character selected from the Options tab of the stream properties dialog box. Otherwise, select either Period(,) or Comma(,) to read all data in this dialog box using the chosen character as the decimal separator.

**Field.** All fields you have defined for this data file are listed here. There are two ways to define fields:
- Specify fields interactively using the data preview pane above.
- Specify fields manually by adding empty field rows to the table below. Click the button to the right of the fields pane to add new fields. Then, in the empty field, enter a Field name, a Start position and a Length. These options will automatically add arrows to the data preview pane, which can be easily adjusted.

To remove a previously defined field, select the field in the list and click the red delete button.

**Start.** Specify the position of the first character in the field. For example, if the second field of a record begins on the sixteenth character, you would enter 16 as the starting point.

**Length.** Specify how many characters are in the longest value for each field. This determines the cutoff point for the next field.

**Strip lead and trail spaces.** Select to discard leading and trailing spaces in strings on import.

**Invalid characters.** Select Discard to remove invalid characters from the data input. Select Replace with to replace invalid characters with the specified symbol (one character only). Invalid characters are null (0) characters or any character that does not exist in the server's encoding.
**Lines to scan for type.** Specify how many lines to scan for specified data types.

At any point while working in this dialog box, click **Refresh** to reload fields from the data source. This is useful when altering data connections to the source node or when working between tabs on the dialog box.

**Setting Data Storage for Text Fields**

The options on the Data tab, common to both Fixed File and Variable File source nodes, allow you to change the storage type for the data fields read into Clementine. In earlier versions of Clementine, data storage was editable only by manipulating the data type. In this release, you can manipulate data storage and data type separately.

- **Data storage** describes the way data is stored in a field. For example, a field with values 1 and 0 is an integer. Other storage types used in Clementine are Real, String, Time, Date, and Timestamp. The Data tab allows you to change the data storage type. You can also convert storage for a field using a variety of conversion functions, such as `to_string` and `to_integer`, in a Filler node. These functions are also available from the Derive node for temporary conversion during a derive calculation.

- **Data type** is a way of describing the intended use of the data in a given field. It is frequently referred to as **usage type**. For example, you may want to set the type for an integer field with values 1 and 0 to a flag. This usually indicates that 1=True and 0=False. When preparing and modeling data, it is often critical that you know the data type for fields of interest. The Types tab in all source nodes allows you to alter the data type. For more information, see “Setting Data Types in the Source Node” on page 119.

**Using the Data Storage Table**

Using the table available on the Data tab, you can perform the following tasks related to data storage.

- Use the **Fields** column to view fields for the current data set.
- Select the check box in the *Override* column to view the current storage type and override if desired.
- Use the *Storage* column to access a drop-down list of storage types for each field. Select a storage type from the list to override the existing storage type.

**Figure 5-4**
*Overriding storage type for a data field*
Additional Options

Several other options can be specified using the Data tab:

- To view storage settings for data that is no longer connected through the current node (train data, for example), select View unused field settings. You can clear the legacy fields by clicking Clear.

- At any point while working in this dialog box, click Refresh to reload fields from the data source. This is useful when altering data connections to the source node or when working between tabs on the dialog box.

Database Node

If you have ODBC (Open Database Connectivity), you can import data from a variety of other packages, including Excel, MS Access, dBASE, SAS (NT version only), Oracle, and Sybase, using the ODBC source node. For information about installing ODBC drivers, see the documentation included on the Clementine CD-ROM. You should begin by reading Getting Started with SPSS Data Access Technology.pdf in the Installation Documents folder.

Use the following general steps to access data from a database:

- In the Database node dialog box, connect to a database using Table mode or SQL Query mode.

- Select a table from the database.

- Using the tabs in the Database node dialog box, you can alter usage types and filter data fields.

These steps are described in more detail in the next several topics.

Setting Database Node Options

You can use the options on the Data tab of the Database node dialog box to gain access to a database and read data from the selected table.
**Figure 5-5**
*Loading data by selecting a table*

![Database Interface](image)

**Mode.** Select Table to connect to a table using the dialog box controls. Select SQL Query to query the database selected below using SQL.

**Data source.** For both Table and SQL Query modes, you can enter a name in the Data source field or select Add new database connection from the drop-down list.

The following options are used to connect to a database and select a table using the dialog box:

**Table name.** If you know the name of the table you would like to access, enter it in the Table name field. Otherwise, click the Select button to open a dialog box listing available tables.

**Strip lead and trail spaces.** Select options for discarding leading and trailing spaces in strings.

**Quote table and column names.** Specify whether you want table and column names to be enclosed in quotation marks when queries are sent to the database (if, for example, they contain spaces or punctuation).
The As needed option will quote table and field names only if they include nonstandard characters. Nonstandard characters include non-ASCII characters, space characters, and any non-alphanumeric character other than a full stop (.).

Select Never if you never want table and field names quoted.

Select Always if you want all table and field names quoted.

**Adding a Database Connection**

In order to open a database, you first have to select the data source to which you want to connect. On the Data tab, select Add new database connection from the Data source drop-down list. This opens the Database Connections dialog box.

**Data sources.** Lists the available data sources. Be sure to scroll down if you do not see the desired database. Once you have selected a data source and entered any passwords, click Connect. Click Refresh to update the list.

**User name.** If the data source is password protected, enter your user name.

**Password.** If the data source is password protected, enter your password.
Connections. Shows currently connected databases. To remove connections, select one from the list and click Remove.

Once you have completed your selections, click OK to return to the main dialog box and select a table from the currently connected database.

Selecting a Database Table

After you have connected to a data source, you can choose to import fields from a specific table or view. From the Data tab of the Database dialog box, you can either enter the name of a table in the Table name field or click Select to open a dialog box listing the available tables and views.

Figure 5-7
Selecting a table from the currently connected database

Show table owner. Select if a data source requires that the owner of a table must be specified before you can access the table. Deselect this option for data sources that do not have this requirement.

Note: SAS and Oracle databases usually require you to show the table owner.

Tables/Views. Select the table or view to import.
Show. Lists the columns in the data source to which you are currently connected. Click one of the following options to customize your view of the available tables:

- Click User Tables to view ordinary database tables created by database users.
- Click System Tables to view database tables owned by the system (such as tables that provide information about the database, like details of indexes). This option is necessary to view the tabs used in Excel databases.
- Click Views to view virtual tables based on a query involving one or more ordinary tables.
- Click Synonyms to view synonyms created in the database for any existing tables.

Querying the Database

Once you have connected to a data source, you can choose to import fields using an SQL query. From the main dialog box, select SQL Query as the connection mode. This adds a query editor window in the dialog box. Using the query editor, you can create or load an SQL query whose result set will be read into the data stream. To cancel and close the query editor window, select Table as the connection mode.
Figure 5-8
Loading data using SQL queries

**Load Query.** Click to open the file browser, which you can use to load a previously saved query.

**Save Query.** Click to open the Save Query dialog box, which you can use to save the current query.

**Import Defaults.** Click to import an example SQL SELECT statement constructed automatically using the table and columns selected in the dialog box.

**Clear.** Clear the contents of the work area. Use this option when you want to start over.

**SPSS Import Node**

You can use the SPSS Import node to read data directly from a saved SPSS file (.sav). This format is now used to replace the Clementine cache file from earlier versions of Clementine. If you would like to import a saved cache file, you should use the SPSS Import node.
**Import file.** Specify the name of the file. You can enter a filename and click or tab to the ellipsis button (…) to select a file. The file path is shown once you have selected a file.

**Use variable labels.** Select if you want to use the descriptive variable labels from the .sav file rather than the short field names. This option is deselected by default, meaning that long names from SPSS are not read into Clementine and are lost upon export.

**Use value labels.** Select if you want to use the value labels from the .sav file rather than the numerical or symbolic codes used to represent the values. For example, selecting this option for data with a gender field whose values 1 and 2 actually represent male and female, respectively, will convert the field to a string and import male and female as the actual values.

It is important to consider missing values in your SPSS data before selecting this option. For example, if a numeric field uses labels only for missing values (0=No Answer, –99=Unknown), then selecting the option above will import only the value
labels *No Answer* and *Unknown* and will convert the field to a string. In such cases, you should import the values themselves and set missing values in a Type node.

**SAS Import Node**

The SAS Import node allows you to bring SAS data into your data mining session. You can import four types of files:

- SAS for Windows/OS2 (*sd2*)
- SAS for UNIX (*ssd*)
- SAS Transport File (*tpt*)
- SAS version 7/8 (*sas7bdat*)

When the data are imported, all variables are kept and no variable types are changed. All cases are selected.

*Figure 5-10*  
Importing a SAS file
Chapter 5

Setting Options for the SAS Import Node

**Import.** Select which type of SAS file to transport. You can choose SAS for Windows/OS2 (.sd2), SAS for UNIX (.SSD), SAS Transport File (.tpt), or SAS Version 7/8 (.sas7bdat).

**Import file.** Specify the name of the file. You can enter a filename or click the ellipsis button (…) to browse to the file's location.

**Member.** Select a member to import from the SAS transport file selected above. You can enter a member name or click Select to browse through all members in the file.

**Read user formats from a SAS data file.** Select to read user formats. SAS files store data and data formats (such as variable labels) in different files. Most often, you will want to import the formats as well. If you have a large data set, however, you may want to deselect this option to save memory.

**Format file.** If a format file is required, this text box is activated. You can enter a filename or click the ellipsis button (…) to browse to the file's location.

**Use variable label headings.** Select to use the descriptive variable labels from the SAS format file rather than the short field names. This option is deselected by default.

User Input Node

The User Input node provides an easy way for you to create synthetic data—either from scratch or by altering existing data. This is useful, for example, when you want to create a test data set for modeling.

**Creating Data from Scratch**

The User Input node is available from the Sources node palette and can be added directly to the stream canvas.

- Click the Sources tab of the nodes palette.
- Drag and drop or double-click to add the User Input node to the stream canvas.
- Double-click to open its dialog box and specify fields and values.
Note: User Input nodes that are selected from the Sources palette will be completely blank, with no fields and no data information. This enables you to create synthetic data entirely from scratch.

Generating Data from an Existing Data Source

You can also generate a User Input node from any non-terminal node in the stream:

- Decide at which point in the stream you want to replace a node.
- Right-click on the node that will feed its data into the User Input node and select Generate User Input Node from the menu.
- The User Input node appears with all downstream processes attached to it, replacing the existing node at that point in your data stream. When generated, the node inherits all of the data structure and field type information (if available) from the metadata.

Note: If data have not been run through all nodes in the stream, then the nodes are not fully instantiated, meaning that storage and data values may not be available when replacing with a User Input node.
Setting Options for the User Input Node

The dialog box for a User Input node contains several tools that you can use to enter values and define the data structure for synthetic data. For a generated node, the table on the Data tab contains field names from the original data source. For a node added from the Sources palette, the table is blank. Using the table options, you can perform the following tasks:

- Add new fields using the add a new field button at the right of the table.
- Rename existing fields.
- Specify data storage for each field.
- Specify values.
**Entering Data and Defining Storage**

For each field, you can specify values or insert values from the original data set using the value picker button to the right of the table. See the rules described below for more information on specifying values. You can also choose to leave the field blank; fields left blank are filled with the system null ($null$).

You must also select a storage type for each field in the synthetic data. Generated User Input nodes may already contain this information, garnered from the source node if instantiated. An uninstantiated node does not contain storage or usage type information.

*Note:* Storage is different from a data type that specifies the usage of a field in Clementine. For more information, see “Data Storage versus Data Type” in Chapter 7 on page 154.

**Figure 5-12**  
*Specifying storage type for fields in a generated User Input node*
Rules for Specifying Values

For symbolic fields, you should leave spaces between multiple values, such as:

**HIGH MEDIUM LOW**

For numeric fields, you can either enter multiple values in the same manner (listed with spaces between):

10 12 14 16 18 20

Or you can specify the same series of numbers by setting its limits (10, 20) and the steps in between (2). Using this method, you would type:

10,20,2

These two methods can be combined by embedding one within the other, such as:

1 5 7 10,20,2 21 23

This entry will produce the following values:

1 5 7 10 12 14 16 18 20 21 23

When you execute a stream, data is read from the fields specified in the User Input node. For multiple fields, one record will be generated for each possible combination of field values. For example, the following entries will generate the records listed in the table below.

- **Age.** 30,60,10
- **BP.** LOW
- **Cholesterol.** NORMALHIGH
- **Drug.** (left blank)

<table>
<thead>
<tr>
<th>Age</th>
<th>BP</th>
<th>Cholesterol</th>
<th>Drug</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>LOW</td>
<td>NORMAL</td>
<td>$null$</td>
</tr>
<tr>
<td>30</td>
<td>LOW</td>
<td>HIGH</td>
<td>$null$</td>
</tr>
</tbody>
</table>
### Common Source Node Tabs

The following options can be specified for all source nodes by clicking the corresponding tab:

- **Data tab.** Used to change the default storage type.
- **Types tab.** Used to set data types. This tab offers the same functionality as the Type node.
- **Filter tab.** Used to eliminate or rename data fields. This tab offers the same functionality as the Filter node.
- **Annotations tab.** Used for all nodes in Clementine, this tab provides options to rename nodes, provide a custom tooltip, and store a lengthy annotation. For more information, see “Annotating Nodes” in Chapter 4 on page 70.

### Setting Data Types in the Source Node

Information about data typing is available from both source and Type nodes. The functionality is similar in both nodes. For more information, see “Type Node” in Chapter 7 on page 148. Using the Types tab in the source node, you can specify a number of important properties of fields:

- **Type.** Used to describe characteristics of the data in a given field. If all details of a field are known, it is said to be **fully instantiated.** The type of a field is different from the storage of a field, which indicates whether data is stored as string, integer, real, date, time, or timestamp. For more information, see “Setting Data Storage for Text Fields” on page 104.

---

<table>
<thead>
<tr>
<th>Age</th>
<th>BP</th>
<th>Cholesterol</th>
<th>Drug</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>LOW</td>
<td>NORMAL</td>
<td>$null$</td>
</tr>
<tr>
<td>40</td>
<td>LOW</td>
<td>HIGH</td>
<td>$null$</td>
</tr>
<tr>
<td>50</td>
<td>LOW</td>
<td>NORMAL</td>
<td>$null$</td>
</tr>
<tr>
<td>50</td>
<td>LOW</td>
<td>HIGH</td>
<td>$null$</td>
</tr>
<tr>
<td>60</td>
<td>LOW</td>
<td>NORMAL</td>
<td>$null$</td>
</tr>
<tr>
<td>60</td>
<td>LOW</td>
<td>HIGH</td>
<td>$null$</td>
</tr>
</tbody>
</table>
Chapter 5

- **Direction.** Used to tell Modeling nodes whether fields will be **Input** (predictor fields) or **Output** (predicted fields) for a machine-learning process. **Both** and **None** are also available directions.

- **Missing values.** Used to specify which values will be treated as **blanks**.

- **Value checking.** In the **Check** column, you can set options to ensure that field values conform to the specified range.

- **Instantiation options.** Using the **Values** column, you can specify options for reading data values from the data set or click **Specify** to open another dialog box for setting values.

**Figure 5-13**
*Types tab options*

Several other options can be specified using the Types tab:

- Using the tools menu button, you can choose to **Ignore Unique Fields** once a Type node has been instantiated (either through your specifications, reading values, or executing the stream). Ignoring unique fields will automatically ignore fields with only one value.
Using the tools menu button, you can choose to Ignore Large Sets once a Type node has been instantiated. Ignoring large sets will automatically ignore sets with a large number of members.

Using the tools menu button, you can generate a Filter node to discard selected fields.

Using the sunglasses toggle buttons, you can set the default for all fields to Read or Pass. The Types tab in the source node passes fields by default, while the Type node itself reads fields by default.

Using the context menu, you can choose to Copy attributes from one field to another. For more information, see “Copying Type Attributes” in Chapter 7 on page 163.

Using the View unused field settings option, you can view type settings for fields that are no longer present in the data or were once connected to this Type node. This is useful when reusing a Type node for train and test data sets.

Using the Clear Values button, you can clear changes to field values made in this node and reread values from the data source. If you have made no alterations in this source node, then pressing Clear Values will make no field changes, effectively setting the Values column options to Pass.

Using the Clear All Values button, you can reset values for all fields read into the node. This option effectively sets the Values column to Read for all fields.

For more information, see “Setting Data Types in the Type Node” in Chapter 7 on page 151.

When to Instantiate at the Source Node

There are two ways you can learn about the data storage and values of your fields. This instantiation can occur at either the source node, when you first bring data into Clementine, or by inserting a Type node into the data stream.

Instantiating at the source node is useful when:

- The data set is small.
- You plan to derive new fields using the Expression Builder (instantiating makes field values available from the E-Builder).
Generally, if your data set is not very large and you do not plan to add fields later in the stream, instantiating at the source node is the most convenient method.

**Filtering Fields from the Source Node**

Using the Filter tab on a source node dialog box allows you to exclude fields from downstream operations based upon your initial examination of the data. This is useful, for example, if there are duplicate fields in the data or if you are already familiar enough with the data to exclude irrelevant fields.

**Figure 5-14**
*Filtering fields from the source node.*

- **Field.** Displays the input fields from currently connected data sources.
- **Filter.** Displays the filter status of all input fields. Filtered fields include a red X in this column indicating that this field will not be passed downstream. Click in the *Filter* column for a selected field to turn filtering on and off. You can also
select options for multiple fields simultaneously using the Shift-click method of selection.

- **Field.** Displays the fields as they leave the Filter node. Duplicate names are displayed in red. You can edit field names by clicking in this column and entering a new name. Or you can remove fields by clicking in the *Filter* column to disable duplicate fields.

All columns in the table above can be sorted by clicking on the column header.

**View current fields.** Select to view fields for data sets actively connected to the Filter node. This option is selected by default and is the most common method of using Filter nodes.

**View unused field settings.** Select to view fields for data sets that were once (but are no longer) connected to the Filter node. This option is useful when copying Filter nodes from one stream to another or saving and reloading Filter nodes.

The Filter menu at the top of this dialog box (available from the filter button) helps you to perform operations on multiple fields simultaneously. You can choose to:

- Remove all fields.
- Include all fields.
- Toggle all fields.
- Remove duplicates.
- Truncate field names.
- Use input field names.
- Set the default filter state.

You can also use the arrow toggle buttons to include all fields or discard all fields at once. This is useful for large data sets where only a few fields are to be included downstream.
Record Operations Nodes

Overview of Record Operations

Record operations nodes are used to make changes to the data set at the record level. These operations are important during the Data Understanding and Data Preparation phases of data mining because they allow you to tailor the data to your particular business need.

For example, based on the results of the data audit conducted using the Data Audit node from the Output palette, you might decide that you'd like customer purchase records for the past three months to be merged. Using a Merge node from the Record Ops palette, you can merge records based on the values of a key field, such as Customer ID. In contrast, for example, you may discover that a database containing information about Web-site hits is unmanageable with over one million records. Using Sample nodes, you can select a subset of data for use in modeling.

Figure 6-1
Record Ops palette

The Record Ops palette contains the following nodes:

- Select
- Sample
- Balance
- Aggregate
- Sort
Chapter 6

- Merge
- Append
- Distinct

Many of the nodes in the Record Ops palette require you to use a CLEM expression. You can use the CLEM Expression Builder to create such expressions automatically, or, if you are familiar with the Clementine Language for Expression Manipulation (CLEM), you can type an expression in the field. To access the Expression Builder, click the button located near all expression fields.

**Figure 6-2**
Expression Builder button

---

**Select Node**

You can use Select nodes to select or discard a subset of records from the data stream based on a specific condition, such as \(BP \text{ (blood pressure)} \text{=} "HIGH"\).

**Figure 6-3**
Select node dialog box

**Mode.** Specifies whether records that meet the condition will be included or excluded from the data stream.
Record Operations Nodes

- **Include.** Select to include records that meet the selection condition.
- **Discard.** Select to exclude records that meet the selection condition.

**Condition.** Displays the selection condition that will be used to test each record, which you specify using a CLEM expression. Either enter an expression in the window or use the Expression Builder by clicking the calculator (Expression Builder) button to the right of the window.

Select nodes are also used to choose a proportion of records. Typically, you would use a different node, the Sample node, for this operation. However, if the condition that you want to specify is more complex than the parameters provided, you can create your own condition using the Select node. For example, you can create a condition such as:

BP = "HIGH" and random(10) <= 4

This will select approximately 40% of the records showing high blood pressure and pass those records downstream for further analysis.

**Sample Node**

You can use Sample nodes to specify a limit on the number of records passed to the data stream or to specify a proportion of records to discard. You may want to sample the original data for a variety of reasons, such as:

- Increasing the performance of the data mining tool.
- Paring down a large data set, such as one with millions of records. Using Sample nodes, you can pass a random sample to generate a model that is usually as accurate as one derived from the full data set.
- Training a neural network. You should reserve a sample for training and a sample for testing.
Setting Options for the Sample Node

Mode. Select whether to pass (include) or discard (exclude) records for the following modes:

- **Pass sample.** Select to **include** in the data stream the sample that you specify below. For example, if you set the mode to Pass sample and set the 1-in-n option to 5, then every fifth record will be included in the data stream up to the maximum sample size.

- **Discard sample.** Select to **exclude** the sample that you specify from the data stream. For example, if you set the mode to Discard sample and set the 1-in-n option to 5, then every fifth record will be discarded (excluded) from the data stream.

Sample. Select the method of sampling from the following options:

- **First.** Select to use contiguous data sampling. For example, if the maximum sample size is set to 10000, then the first 10,000 records will either be passed on to the data stream (if the mode is Pass sample) or discarded (if the mode is Discard sample).
Record Operations Nodes

- **1-in-n.** Select to sample data by passing or discarding every $n$th record. For example, if $n$ is set to 5, then every fifth record will either be passed to the data stream or discarded, depending on the mode selected above.

- **Random %.** Select to sample a random percentage of the data. For example, if you set the percentage to 20, then 20% of the data will either be passed to the data stream or discarded, depending on the mode selected above. Use the field to specify a sampling percentage. You can also specify a seed value using the Set random seed control below.

**Maximum sample size.** Specify the largest sample to be included or discarded from the data stream. This option is redundant and therefore disabled when First and Include are selected above.

**Set random seed.** When Random % is selected above, you can use this control to set a random seed and specify the seed value. Specifying a seed value allows you to reproduce the same list of randomly selected records if needed. Click the Generate button to automatically generate a random seed.

**Balance Node**

You can use Balance nodes to correct imbalances in data sets so that they conform to specified test criteria. For example, suppose that a data set has only two values—*low* or *high*—and that 90% of the cases are *low* while only 10% of the cases are *high*. Many modeling techniques have trouble with such biased data because they will tend to learn only the *low* outcome and ignore the *high* one, since it is more rare. If the data are well-balanced with approximately equal numbers of *low* and *high* outcomes, models will have a better chance of finding patterns that distinguish the two groups. In this case, a Balance node is useful for creating a balancing directive that reduces cases with a *low* outcome.

Balancing is carried out by duplicating and then discarding records based on the conditions that you specify. Records for which no condition holds are always passed through. Because this process works by duplicating and/or discarding records, the original sequence of your data is lost in downstream operations. Be sure to derive any sequence-related values before adding a Balance node to the data stream.

*Note:* Balance nodes can be generated automatically from distribution charts and histograms.
Setting Options for the Balance Node

Figure 6-5
Balance node dialog box

Record balancing directives. Lists the current balancing directives. Each directive includes both a factor and a condition that tells the software to “increase the proportion of records by a factor specified where the condition is true.” A factor lower than 1.0 means that the proportion of indicated records will be decreased. For example, if you want to decrease the number of records where drug Y is the treatment drug, you might create a balancing directive with a factor of 0.7 and a condition Drug = "drugY". This directive means that the number of records where drug Y is the treatment drug will be reduced to 70% for all downstream operations.

Note: Balance factors for reduction may be specified to four decimal places. Factors set below 0.0001 will result in an error, since the results do not compute correctly.

- **Create conditions** by clicking the button to the right of the text field. This inserts an empty row for entering new conditions. To create a CLEM expression for the condition, click the Expression Builder button.

- **Delete directives** using the red delete button.

- **Sort directives** using the up and down arrow buttons.
Aggregate Node

Aggregation is a data preparation task frequently used to reduce the size of a data set. Before proceeding with aggregation, you should take time to clean the data, concentrating especially on missing values. Once you have aggregated, potentially useful information regarding missing values may be lost. For more information, see “Overview of Missing Values” in Chapter 9 on page 223.

You can use an Aggregate node to replace a sequence of input records with summary, aggregated output records. For example, you might have a set of input records such as:

<table>
<thead>
<tr>
<th>Age</th>
<th>Sex</th>
<th>Region</th>
<th>Branch</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>M</td>
<td>S</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>45</td>
<td>M</td>
<td>S</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>37</td>
<td>M</td>
<td>S</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>30</td>
<td>M</td>
<td>S</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>44</td>
<td>M</td>
<td>N</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>25</td>
<td>M</td>
<td>N</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>29</td>
<td>F</td>
<td>S</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>41</td>
<td>F</td>
<td>N</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>23</td>
<td>F</td>
<td>N</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>45</td>
<td>F</td>
<td>N</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>33</td>
<td>F</td>
<td>N</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

You can aggregate these records with Sex and Region as key fields. Then choose to aggregate Age with the mode Mean and Sales with the mode Sum. Select Include record count in field in the Aggregate node dialog box and your aggregated output would be:

<table>
<thead>
<tr>
<th>Age</th>
<th>Sex</th>
<th>Region</th>
<th>Sales</th>
<th>RECORD_COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>35.5</td>
<td>F</td>
<td>N</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>34.5</td>
<td>M</td>
<td>N</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>29</td>
<td>F</td>
<td>S</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>33.75</td>
<td>M</td>
<td>S</td>
<td>20</td>
<td>4</td>
</tr>
</tbody>
</table>
Note that fields such as *Branch* are automatically discarded when no aggregate mode is specified.

**Figure 6-6**  
*Aggregate node dialog box*

![Aggregate node dialog box](image)

**Setting Options for the Aggregate Node**

**Key fields.** Lists fields that can be used as keys for aggregation. Both numeric and symbolic fields can be used as keys. If you choose more than one key field, the values will be combined to produce a key value for aggregating records. One aggregated record will be generated for each unique key field. For example, if *Sex* and *Region* are your key fields, each unique combination of *M* and *F* with regions *N* and *S* (four unique combinations) will have an aggregated record. To add a key field, use the Field Chooser button to the right of the window.
**Keys are contiguous.** Select to treat the values for the key fields as equal if they occur in adjacent records.

**Aggregate fields.** Lists the numeric fields whose values will be aggregated as well as the selected modes of aggregation. To add fields to this list, use the Field Chooser button on the right.

**Default mode.** Specify the default aggregation mode to be used for newly added fields. If you frequently use the same aggregation, select one or more modes here and use the Apply to All button on the right to apply the selected modes to all fields listed above. The following aggregation modes are available in Clementine:

- **Sum.** Select to return summed values for each key field combination.
- **Mean.** Select to return the mean values for each key field combination.
- **Min.** Select to return minimum values for each key field combination.
- **Max.** Select to return maximum values for each key field combination.
- **SDev.** Select to return the standard deviation for each key field combination.

**New field name extension.** Select to add a suffix or prefix, such as “1” or “new,” to duplicate aggregated fields. For example, the result of a minimum values aggregation on the field *Age* will produce a field name called *Age_Min_1* if you have selected the suffix option and specified “1” as the extension. **Note:** Aggregation extensions such as _Min or Max_ are automatically added to the new field, indicating the type of aggregation performed. Select Suffix or Prefix to indicate your preferred extension style.

**Include record count in field.** Select to include an extra field in each output record called *Record_Count*, by default. This field indicates how many input records were aggregated to form each aggregate record. Create a custom name for this field by typing in the edit field.

**Note:** System null values are excluded when aggregates are computed, but they are included in the record count. Blank values, on the other hand, are included in both aggregation and record count. To exclude blank values, you can use a Filler node to replace blanks with null values. You can also remove blanks using a Select node.
You can use Sort nodes to sort records into ascending or descending order based on the values of one or more fields. For example, Sort nodes are frequently used to view and select records with the most common data values. Typically, you would first aggregate the data using the Aggregate node and then use the Sort node to sort the aggregated data into descending order of record counts. Displaying these results in a table will allow you to explore the data and to make decisions, such as selecting the records of the top-10 best customers.

**Sort by.** All fields selected to use as sort keys are displayed in a table. A key field works best for sorting when it is numeric.

- **Add fields** to this list using the Field Chooser button on the right.
- **Select an order** by clicking the Ascending or Descending arrow in the table's Order column.
- **Delete fields** using the red delete button.
- **Sort directives** using the up and down arrow buttons.

**Default sort order.** Select either Ascending or Descending to use as the default sort order when new fields are added above.
Merge Node

The function of a Merge node is to take multiple input records and create a single output record containing all or some of the input fields. This is a useful operation when you want to merge data from different sources, such as internal customer data and purchased demographic data. There are two ways to merge data in Clementine:

- **Merge by order** concatenates corresponding records from all sources in the order of input until the smallest data source is exhausted. It is important if using this option that you have sorted your data using a Sort node.

- **Merge using a key field**, such as *Customer ID*, to specify how to match records from one data source with records from the other(s). Several types of joins are possible in Clementine, including inner join, full outer join, partial outer join, and anti-join. For more information, see “Types of Joins” on page 135.

Types of Joins

When using a key field for data merging, it is useful to spend some time thinking about which records will be excluded and which will be included. Clementine offers a variety of joins, which are discussed in detail below.

The two basic types of joins are referred to as inner and outer joins. These methods are frequently used to merge tables from related data sets based on common values of a key field, such as Customer ID. Inner joins allow for clean merging and an output data set that includes only complete records. Outer joins also include complete records from the merged data, but they also allow you to include unique data from one or more input tables.

The types of joins allowed in Clementine are described in greater detail below.
Chapter 6

<table>
<thead>
<tr>
<th>Diagram</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td>An <strong>inner join</strong> includes only records where a value for the key field is common to all input tables. That is, unmatched records will not be included in the output data set.</td>
</tr>
<tr>
<td><img src="image2.png" alt="Diagram" /></td>
<td>A <strong>full outer join</strong> includes all records, both matching and unmatching, from the input tables. Left and right outer joins are referred to as partial outer joins and are described below.</td>
</tr>
<tr>
<td><img src="image3.png" alt="Diagram" /></td>
<td>A <strong>partial outer join</strong> includes all records matched using the key field as well as unmatched records from specified tables. Tables (such as A and B shown here) can be selected for inclusion in the outer join using the Select button on the Merge tab. Partial joins are also called left or right outer joins when only two tables are being merged. Since Clementine allows the merging of more than two tables, we refer to this as a partial outer join.</td>
</tr>
<tr>
<td><img src="image4.png" alt="Diagram" /></td>
<td>An <strong>anti-join</strong> includes only unmatched records for the first input table (Table A shown here). This type of join is the opposite of an inner join and does not include complete records in the output data set.</td>
</tr>
</tbody>
</table>
**Specifying a Merge Method and Keys**

Figure 6-8  
*Using the Merge tab to set merge method options*

---

**Merge Method.** Select either *Order* or *Keys* to specify the method of merging records. Selecting *Keys* activates the bottom half of the dialog box.

- **Order.** Merges records by order such that the *n*th record from each input is merged to produce the *n*th output record. When any record runs out of a matching input record, no more output records are produced. This means that the number of records created is the number of records in the smallest data set.

- **Keys.** Uses a key field, such as *Transaction ID*, to merge records with the same value in the key field. This is equivalent to a database “equi-join.” If a key value occurs more than once, all possible combinations are returned. For example, if records with the same key field value *A* contain differing values *B*, *C*, and
\( D \) in other fields, the merged fields will produce a separate record for each combination of \( A \) with value \( B \), \( A \) with value \( C \), and \( A \) with value \( D \).

*Note:* Null values are not considered identical in the merge-by-key method and will not join.

**Possible keys.** Lists all fields found in all input data sources. Select a field from this list and use the arrow button to add it as a key field used for merging records. More than one key field may be used.

**Keys for merge.** Lists all fields used to merge records from all input data sources based on values of the key fields. To remove a key from the list, select one and use the arrow button to return it to the Possible keys list. When more than one key field is selected, the option below is enabled.

**Combine duplicate key fields.** When more than one key field is selected above, this option ensures that there is only one output field of that name. This option is enabled by default except in the case when streams have been imported from earlier versions of Clementine. When this option is disabled, duplicate key fields must be renamed or excluded using the Filter tab in the Merge node dialog box.

**Include only matching records (inner join).** Select to merge only complete records.

**Include matching and nonmatching records (full outer join).** Select to perform a “full outer join.” This means that if values for the key field are not present in all input tables, the incomplete records are still retained. The undefined value (\$null\$) is added to the key field and included in the output record.

**Include matching and selected nonmatching records (partial outer join).** Select to perform a “partial outer join” of the tables you select in a subdialog box. Click Select to specify tables for which incomplete records will be retained in the merge.

**Include records in the first data set not matching any others (anti-join).** Select to perform a type of “anti-join” where only nonmatching records from the first data set are passed downstream. You can specify the order of input data sets using arrows on the Inputs tab. This type of join does not include complete records in the output data set.

For more information, see “Types of Joins” on page 135.
Selecting Data for Partial Joins

For a partial outer join, you must select the table(s) for which incomplete records will be retained. For example, you may want to retain all records from a Customer table while retaining only matched records from the Mortgage Loan table.

Figure 6-9
Selecting data for a partial or outer join

Outer Join column. In the Outer Join column, select data sets to include in their entirety. For a partial join, overlapping records will be retained as well as incomplete records for data sets selected here. For more information, see “Types of Joins” on page 135.

Filtering Fields from the Merge Node

Merge nodes include a convenient way of filtering or renaming duplicate fields as a result of merging multiple data sources. Click the Filter tab in the dialog box to select filtering options.
The options presented here are nearly identical to those for the Filter node. There are, however, additional options not discussed here that are available on the Filter menu. For more information, see “Filter Node” in Chapter 7 on page 164.

- **Field.** Displays the input fields from currently connected data sources.
- **Tag.** Lists the tag name (or number) associated with the data source link. Click the Inputs tab to alter active links to this Merge node.
- **Source node.** Displays the source node whose data is being merged.
- **Connected node.** Displays the node name for the node that is connected to the Merge node. Frequently, complex data mining requires several merge or append operations that may include the same source node. The connected node name provides a way of differentiating these.

**Filter.** Displays the current connections between input and output field. Active connections show an unbroken arrow. Connections with a red X indicate filtered fields.

**Field.** Lists the output fields after merging or appending. Duplicate fields are displayed in red. Click in the Filter field above to disable duplicate fields.
**View current fields.** Select to view information on fields selected to be used as key fields.

**View unused field settings.** Select to view information on fields that are not currently in use.

### Setting Input Order and Tagging

Using the Inputs tab in the Merge and Append node dialog boxes, you can specify the order of input data sources and make any changes to the tag name for each source.

Figure 6-11

*Using the Inputs tab to specify tags and input order*

<table>
<thead>
<tr>
<th>Tags and order of input datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>transaction</td>
</tr>
<tr>
<td>products</td>
</tr>
</tbody>
</table>

**Tags and order of input data sets.** Select to merge or append only complete records.

- Tag. Lists current tag names for each input data source. Tag names, or tags, are a way of uniquely identifying the data links for the merge or append operation. For example, imagine water from various pipes that is combined at one point and flows through a single pipe. Data in Clementine flows similarly, and the merging point is often a complex interaction between the various data sources. Tags
provide a way of managing the inputs (“pipes”) to a Merge or Append node so that if the node is saved or disconnected, the links remain and are easily identifiable.

When you connect additional data sources to a Merge or Append node, default tags are automatically created using numbers to represent the order in which you connected the nodes. This order is unrelated to the order of fields in the input or output data sets. You can change the default tag by entering a new name in the *Tag* column.

- **Source Node.** Displays the source node whose data is being combined.
- **Connected Node.** Displays the node name for the node that is connected to the Merge or Append node. Frequently, complex data mining requires several merge operations that may include the same source node. The connected node name provides a way of differentiating these.
- **Fields.** Lists the number of fields in each data source.

**View current tags.** Select to view tags that are actively being used by the Merge or Append node. In other words, current tags identify links to the node that have data flowing through. Using the pipe metaphor, current tags are analogous to pipes with existing water flow.

**View unused tag settings.** Select to view tags, or links, that were previously used to connect to the Merge or Append node but are not currently connected with a data source. This is analogous to empty pipes still intact within a plumbing system. You can choose to connect these “pipes” to a new source or remove them. To remove unused tags from the node, click **Clear**. This clears all unused tags at once.
Append Node

You can use Append nodes to concatenate sets of records together. Unlike Merge nodes, which join records from different sources together, Append nodes read and pass downstream all of the records from one source until there are no more. Then the records from the next source are read using the same data structure (number of records and fields, etc.) as the first, or primary, input. When the primary source has more fields than another input source, the system null string ($null$) will be used for any incomplete values.

Append nodes are useful for combining data sets with similar structures but different data. For example, you might have transaction data stored in different files for different time periods, such as a sales data file for March and a separate one for April. Assuming that they have the same structure (the same fields in the same order), the Append node will join them together into one large file, which you can then analyze.

Note: In order to append files, the field types must be similar. For example, a field typed as a Set field can not be appended with a field typed as Real Range.
**Setting Append Options**

**Match fields by.** Select a method to use when matching fields to append.

- **Position.** Select to append data sets based on the position of fields in the main data source. When using this method, your data should be sorted to ensure proper appending.

- **Name.** Select to append data sets based on the name of fields in the input data sets. Also select Match case to enable case sensitivity when matching field names.

**Output Field.** Lists the source nodes that are connected to the Append node. The first node on the list is the primary input source. You can sort the fields in the display by clicking on the column heading. This sorting does not actually reorder the fields in the data set.
Include fields from. Select Main data set only to produce output fields based on the fields in the main data set. The main data set is the first input, specified on the Inputs tab. Select All data sets to produce output fields for all fields in all data sets regardless of whether there is a matching field across all input data sets.

Tag records by including source data set in field. Select to add an additional field to the output file whose values indicate the source data set for each record. Specify a name in the text field. The default field name is Input.

Distinct Node

You can use Distinct nodes to remove duplicate records either by passing the first distinct record to the data stream or by discarding the first record and passing any duplicates to the data stream instead. This operation is useful when you want to have a single record for each item in the data, such as customers, accounts, or products. For example, Distinct nodes can be helpful in finding duplicate records in a customer database or in getting an index of all of the product IDs in your database.

Figure 6-14
Distinct node dialog box

Mode. Specify whether to include or exclude (discard) the first record.
Chapter 6

- **Include.** Select to include the first distinct record in the data stream.
- **Discard.** Select to discard the first distinct record found and pass any duplicate records to the data stream instead. This option is useful for finding duplicates in your data so that you can examine them later in the stream.

**Fields.** Lists fields used to determine whether records are identical.

- **Add fields** to this list using the Field Chooser button on the right.
- **Delete fields** using the red delete button.
Chapter 7

Field Operations Nodes

Field Operations Overview

Based on your findings in the data audit conducted during the initial data exploration, you will often have to select, clean, and construct data. The Field Ops palette contains many nodes useful for this transformation and preparation.

For example, using a Derive node, you can create an attribute, such as length of service, that is not currently represented in the data. Or you can use a Binning node, for example, to automatically recode field values for targeted analysis. Type nodes are frequently used because they allow you to assign data type, values, and direction for each field in the data set. These operations are useful for handling missing values and downstream modeling.

Figure 7-1
Field Ops palette

The Field Ops palette contains the following nodes:

- Type
- Filter
- Derive
- Filler
- Reclassify
- Binning
Several of these nodes can be generated directly from the audit report created by a Data Audit node. For more information, see “Generating Graphs and Nodes from the Audit” in Chapter 14 on page 516.

**Type Node**

Using Type nodes, you can specify a number of important properties of fields:

- **Type.** Used to describe characteristics of the data in a given field. If all of the details of a field are known, it is called **fully instantiated**. The type of a field is different from the storage of a field, which indicates whether data are stored as strings, integers, real numbers, dates, times, or timestamps.

- **Direction.** Used to tell Modeling nodes whether fields will be **Input** (predictor fields) or **Output** (predicted fields) for a machine learning process. **Both** and **None** are also available directions.

- **Missing values.** Used to specify which values will be treated as **blanks**.

- **Value checking.** In the **Check** column, you can set options to ensure that field values conform to the specified range.

- **Instantiation options.** Using the **Values** column, you can specify options for reading data values from the data set, or use the Specify option to open another dialog box for setting values. You can also choose to pass fields without reading their values.
Several other options can be specified using the Type node window:

- Using the tools menu button, you can choose to Ignore Unique Fields once a Type node has been instantiated (either through your specifications, reading values, or executing the stream). Ignoring unique fields will automatically ignore fields with only one value.

- Using the tools menu button, you can choose to Ignore Large Sets once a Type node has been instantiated. Ignoring large sets will automatically ignore sets with a large number of members.

- Using the tools menu button, you can generate a Filter node to discard selected fields.

- Using the sunglasses toggle buttons, you can set the default for all fields to Read or Pass. The Types tab in the source node passes fields by default, while the Type node itself reads values by default.

- Using the Clear Values button, you can clear changes to field values made in this node (non-inherited values) and reread values from upstream operations. This option is useful for resetting changes that you may have made for specific fields upstream.
Using the Clear All Values button, you can reset values for all fields read into the node. This option effectively sets the Values column to Read for all fields. This option is useful to reset values for all fields and reread values and types from upstream operations.

Using the context menu, you can choose to Copy attributes from one field to another. For more information, see “Copying Type Attributes” on page 163.

Using the View unused field settings option, you can view type settings for fields that are no longer present in the data or were once connected to this Type node. This is useful when reusing a Type node for data sets that have changed.

Data Types

The following types describe the way data is used in Clementine:

- **Range.** Used to describe numeric values, such as a range of 0–100 or 0.75–1.25. A range value may be an integer, real number, or date/time.

- **Discrete.** Used for string values when an exact number of distinct values is unknown. This is an uninstantiated data type, meaning that all possible information about the storage and usage of the data is not yet known. Once data has been read, the type will be flag, set, or typeless, depending on the maximum set size specified in the stream properties dialog box.

- **Flag.** Used for data with two distinct values, such as Yes/No or 1, 2. Data may be represented as text, integer, real number, or date/time. *Note:* Date/time refers to three types of storage: time, date, or timestamp.

- **Set.** Used to describe data with multiple distinct values, each treated as a member of a set, such as small/medium/large. In this version of Clementine, a set can have any storage—numeric, string, or date/time. Note that setting a type to Set does not automatically change the values to string.

- **Typeless.** Used for data that does not conform to any of the above types or for set types with too many members. It is useful for cases in which the type would otherwise be a set with many members (such as an account number). When you select Typeless for a field, the direction is automatically set to None. The default maximum size for sets is 250 unique values. This number can be adjusted or disabled in the stream properties dialog box.
**Setting Data Types in the Type Node**

Using Type nodes, you can specify several important properties of fields.

A data **type** is essentially metadata about the particular field, and it describes the way the values of this data are used in Clementine. This metadata is not available automatically. Data types become known, or available, in one of two ways:

- By manually setting the type from the Type node or source node
- By “auto-typing,” or letting the software read the data and determine the type based on the values that it reads

In both cases, the *Type* column of the data types table contains information about existing types for all fields included at this point in the stream. You can alter types and values using this table.

**To use auto-typing:**

The following methods are used to let the software determine usage types for data fields:

- Insert a Type node in the data stream and set the *Values* column to `<Read>` or `<Read +>`.
- Using the Types tab of a source node, set the *Values* column to `<Read>` for all fields. This will make metadata available to all nodes downstream. You can quickly set all fields to `<Read>` or `<Pass>` using the sunglasses buttons on the dialog box.
- Attach a terminal node to the Type node in a data stream. Executing the terminal node runs data through the stream, giving the software a chance to learn about the values in your data and thus making the type known. You can also use the *Read Values* button to read values from the data source immediately.

**To manually set the type for a field:**

- Select a field in the table.
- From the drop-down list in the *Type* column, select a type for the field.
- Alternatively, you can use Ctrl-A or the Ctrl-click method to select multiple fields before using the drop-down list to select a type.
Chapter 7

Note: Selecting a type will not affect the data storage used for this field. You can alter data storage using the Data tab in the Variable File and Fixed File node dialog boxes or using the conversion functions available from the Filler node.

Figure 7-3
Manually setting types

What Is Instantiation?

Instantiation is the process of reading or specifying information, such as storage type and values for a data field. In order to optimize system resources, instantiating in Clementine is a user-directed process—you tell the software to read values by specifying options on the Types tab in a source node or by running data through a Type node.

- Data with unknown types are also referred to as uninstantiated. Data whose storage type and values are unknown are displayed in the Type column of the Types tab as <Default>. 
When you have some information about a field's storage, such as string or numeric, the data is called **partially instantiated**. Discrete or Range are partially instantiated types. For example, Discrete specifies that the field is symbolic, but you don't know whether it is a set or a flag type.

When all of the details about a type are known, including the values, a **fully instantiated** type—set, flag, range—is displayed in this column. *Note:* The range type is used for both partially instantiated and fully instantiated data fields. Ranges can be either integers or real numbers.

During the execution of a data stream with a Type node, uninstantiated types immediately become partially instantiated, based on the initial data values. Once all of the data have passed through the node, all data become fully instantiated unless values were set to `<Pass>`. If execution is interrupted, the data will remain partially instantiated. Once the Types tab has been instantiated, the values of a field are static at that point in the stream. This means that any upstream changes will not affect the values of a particular field, even if you reexecute the stream. To change or update the values based on new data or added manipulations, you need to edit them in the Types tab itself or set the value for a field to `<Read>` or `<Read +>`.

### When to Instantiate at the Type Node

There are two ways you can learn about the storage type and values of your data fields. This **instantiation** can occur at either the source node when you first bring data into Clementine or by inserting a Type node into the data stream.

Instantiating at the Type node is useful when:

- The data set is large, and the stream filters a subset prior to the Type node.
- Data have been filtered in the stream.
- Data have been merged or appended in the stream.
- New data fields are derived during processing.

Generally, if your data set is not very large and you do not plan to add fields later in the stream, instantiating at the source node is the most convenient method. The Type node provides additional flexibility for large data sets and compatibility with earlier versions of Clementine.
Data Storage versus Data Type

The type of a field is different than the storage of a field, which indicates whether data is stored as a string, integer, real number, date, time, or timestamp. Storage can be changed at the source node for Fixed File and Variable File nodes or using a conversion function, such as to_integer, in a Filler node. Data type describes the usage of the data fields in Clementine and does not affect storage.

Reading Data Values

Using the Values column of the data types table, you can select auto-typing options or specify types and values in a separate dialog box.

Figure 7-4
Selecting methods for reading, passing, or specifying data values
The options available from this drop-down list provide the following instructions for auto-typing:

<table>
<thead>
<tr>
<th>Option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Read&gt;</td>
<td>Data will be read when the node is executed.</td>
</tr>
<tr>
<td>&lt;Read+&gt;</td>
<td>Data will be read and appended to the current data (if any exists).</td>
</tr>
<tr>
<td>&lt;Pass&gt;</td>
<td>No data is read.</td>
</tr>
<tr>
<td>&lt;Current&gt;</td>
<td>Keep current data values.</td>
</tr>
<tr>
<td>Specify...</td>
<td>A separate dialog box is launched for you to specify values and type options.</td>
</tr>
</tbody>
</table>

Executing a Type node or clicking Read Values will auto-type and read values from your data source based on your selection. These values can also be specified manually using the Specify option or by double-clicking a cell in the Field column.

Once you have made changes for fields in the Type node, you can reset value information using the following buttons on the dialog box toolbar:

- Using the Clear Values button, you can clear changes to field values made in this node (non-inherited values) and reread values from upstream operations. This option is useful for resetting changes that you may have made for specific fields upstream.

- Using the Clear All Values button, you can reset values for all fields read into the node. This option effectively sets the Values column to Read for all fields. This option is useful to reset values for all fields and reread values and types from upstream operations.

**Using the Values Dialog Box**

This dialog box contains several options for reading, specifying, and handling values for the selected field. Many of the controls are common to all types of data. These common controls are discussed here.
**Figure 7-5**
*Setting options for data values*

![Drug Values dialog box](image)

**Type.** Displays the currently selected type. You can change the type to reflect the way that you intend to use data in Clementine. For instance, if a field called `day_of_week` contains numbers representing individual days, you may want to change this type to a set in order to create a distribution node that examines each category individually.

**Storage.** Displays the storage type if known. Storage types are unaffected by the usage type (range, set, flag) that you choose for work in Clementine. To alter the storage type, you can use the Data tab in Fixed File and Variable File source nodes or a conversion function in a Filler node.

**Values.** Select a method to determine values for the selected field. Selections that you make here override any selections that you made earlier from the Values column of the Type node dialog box. Choices for reading values include:

- **Read from data.** Select to read values when the node is executed. This option is the same as `<Read>`.
- **Pass.** Select not to read data for the current field. This option is the same as <Pass>.

- **Specify values.** Used in conjunction with value checking, this option allows you to specify values based on your knowledge of the current field. This option activates unique controls for each type of field. These options are covered individually in subsequent topics. *Note:* You cannot specify values for a typeless or <Default> field type.

**Extend values from data.** Select to append the current data with the values that you enter here. For example, if field_1 has a range from (0,10), and you enter a range of values from (8,16), the range is extended by adding the 16, without removing the original minimum. The new range would be (0,16). Choosing this option automatically sets the auto-typing option to <Read+>.

**Check values.** Select a method of coercing values to conform to the specified range, flag, or set values. This option corresponds to the Check column in the Type node dialog box, and settings made here override those in the dialog box. Used in conjunction with the Specify values option, value checking allows you to conform values in the data with expected values. For example, if you specify values as 1, 0 and then use the Discard Check option, you can discard all records with values other than 1 or 0.

**Define blanks.** Select to activate the controls below that enable you to declare missing values or blanks in your data. You can specify system nulls (displayed in the data as $null$) and white space (values with no visible characters) as blanks. By default, selecting Define Blanks enables null-checking for all fields and white space for string or unknown fields. You can also use the Missing values table to define specific values (such as 99 or 0) as blanks.

*Note:* To code blanks as undefined or $null$, you should use the Filler node.

**Description.** Use this text box to enter any comments regarding data fields. These are used as tooltips in a variety of locations, such as the Expression Builder.

### Specifying Values for a Range

The range type is used for numeric fields. There are three storage types for range type nodes:

- Real
Chapter 7

- Integer
- Date/Time

The same dialog box is used to edit all types of range nodes; however, the different storage types are displayed as reference.

Figure 7-6
Options for specifying a range of values

![Age Values dialog box]

Lower. Specify a lower limit for the range field values.

Upper. Specify an upper limit for the range field values.

**Specifying Values for a Set**

Set field types indicate that the data values are used discretely as a member of the set. The storage types for a set can be string, integer, real number, or date/time.
Values. Used in conjunction with value checking, this option allows you to specify values based on your knowledge of the current field. Using this table, you can enter expected values for the field and check the data set's conformity to these values using the Check values drop-down list. Using the arrow and delete buttons, you can modify existing values as well as reorder or delete values.

Specifying Values for a Flag

Flag fields are used to display data that have two distinct values. The storage types for flags can be string, integer, real number, or date/time.
**Figure 7-8**
Options for specifying flag field values

True. Specify a flag value for the field when the condition is met.

False. Specify a flag value for the field when the condition is not met.

**Checking Type Values**

Turning on the Check option for each field examines all values in that field to determine whether they comply with the current type settings or the values that you have specified in the specify values dialog box. This is useful for cleaning up data sets and reducing the size of a data set within a single operation.
The setting of the Check column in the Type node dialog box determines what happens when a value outside of the type limits is discovered. To change the Check settings for a field, use the drop-down list for that field in the Check column. To set the Check settings for all fields, click in the Field column and press Ctrl-A. Then use the drop-down list for any field in the Check column.

The following Check settings are available:

**None.** Values will be passed through without checking. This is the default setting.

**Nullify.** Change values outside of the limits to the system null ($null$).

**Coerce.** Fields whose types are fully instantiated will be checked for values that fall outside the specified ranges. Unspecified values will be converted to a legal value for that type using the following rules:

- For flags, any value other than the true and false value is converted to the false value.
- For sets, any unknown value is converted to the first member of the set's values.
- Numbers greater than the upper limit of a range are replaced by the upper limit.
Numbers less than the lower limit of a range are replaced by the lower limit.
- Null values in a range are given the midpoint value for that range.

**Discard.** When illegal values are found, the entire record is discarded.

**Warn.** The number of illegal items is counted and reported in the stream properties dialog box when all of the data have been read.

**Abort.** The first illegal value encountered terminates the execution of the stream. The error is reported in the stream properties dialog box.

### Setting Field Direction

Specifying the direction of a field provides useful information for Modeling nodes because they tell the model engine in which direction fields will be used. You can set a direction by clicking in the **Direction** column for a particular field.

**Figure 7-10**

*Setting Direction options for the Type node*

Directions can be set as:

**In.** The field will be used as an input to machine learning (a predictor field).
**Out.** The field will be used as an output or target for machine learning (one of the fields that the model will try to predict).

**Both.** The field will be used as both an input and an output by the GRI and Apriori nodes. All other modeling nodes will ignore the field.

**None.** The field will be ignored by machine learning. Fields that have been set to Typeless are automatically set to None in the Direction column.

---

**Copying Type Attributes**

You can easily copy the attributes of a type, such as values, checking options, and missing values from one field to another:

1. Right-click on the field whose attributes you want to copy.
2. From the context menu, choose Copy.
3. Right-click on the field(s) whose attributes you want to change.
4. From the context menu, choose Paste Special. *Note:* You can select multiple fields using the Ctrl-click method or by using the Select Fields option from the context menu.

A new dialog box opens, from which you can select the specific attributes that you want to paste. If you are pasting into multiple fields, the options that you select here will apply to all target fields.

**Paste the following attributes.** Select from the list below to paste attributes from one field to another.

- **Type.** Select to paste the type.
- **Values.** Select to paste the field values.
- **Missing.** Select to paste missing value settings.
- **Check.** Select to paste value checking options.
- **Direction.** Select to paste the direction of a field.
Filter Node

Filter nodes have three functions:

- To filter or discard fields from records that pass through them. For example, as a medical researcher, you may not be concerned about the potassium level (field level data) of patients (record level data); therefore, you can filter out the K (potassium) field.
- To rename fields.
- To map fields from one source node to another. For more information, see “Mapping Data Streams” in Chapter 4 on page 89.

Figure 7-11
Setting Filter node options
Setting Filtering Options

The table used in the Filter tab shows the name of each field as it comes into the node as well as the name of each field as it leaves. You can use the options in this table to rename or filter out fields that are duplicates or are unnecessary for downstream operations.

- **Field.** Displays the input fields from currently connected data sources.

- **Filter.** Displays the filter status of all input fields. Filtered fields include a red X in this column, indicating that this field will not be passed downstream. Click in the Filter column for a selected field to turn filtering on and off. You can also select options for multiple fields simultaneously using the Shift-click method of selection.

- **Field.** Displays the fields as they leave the Filter node. Duplicate names are displayed in red. You can edit field names by clicking in this column and entering a new name. Or, remove fields by clicking in the Filter column to disable duplicate fields.

All columns in the table can be sorted by clicking on the column header.

**View current fields.** Select to view fields for data sets actively connected to the Filter node. This option is selected by default and is the most common method of using Filter nodes.

**View unused field settings.** Select to view fields for data sets that were once but are no longer connected to the Filter node. This option is useful when copying Filter nodes from one stream to another or when saving and reloading Filter nodes.

The filter menu at the top of this dialog box (available from the filter button) helps you to perform operations on multiple fields simultaneously.
You can choose to:

- Remove all fields.
- Include all fields.
- Toggle all fields.
- Remove duplicates. *Note:* Selecting this option removes all occurrences of the duplicate name, including the first one.
- Truncate field names.
- Use input field names.
- Set the default filter state.

You can also use the arrow toggle buttons at the top of the dialog box to include all fields or discard all fields at the same time. This is useful for large data sets where only a few fields are to be included downstream.
Truncating Field Names

Using the options from the filter menu button, you can choose to truncate field names.

**Maximum length.** Specify a number of characters to limit the length of field names.

**Number of digits.** If field names, when truncated, are no longer unique, they will be further truncated and differentiated by adding digits to the name. You can specify the number of digits used. Use the arrow buttons to adjust the number.

For example, the table below illustrates how field names in a medical data set are truncated using the default settings (Maximum Length = 8 and Number of Digits = 2).

<table>
<thead>
<tr>
<th>Field Names</th>
<th>Truncated Field Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient Input 1</td>
<td>Patien01</td>
</tr>
<tr>
<td>Patient Input 2</td>
<td>Patien02</td>
</tr>
<tr>
<td>Heart Rate</td>
<td>HeartRat</td>
</tr>
<tr>
<td>BP</td>
<td>BP</td>
</tr>
</tbody>
</table>

Derive Node

One of the most powerful features in Clementine is the ability to modify data values and derive new fields from existing data. During lengthy data mining projects, it is common to perform several derivations, such as extracting a customer ID from a string of Web-log data or creating a customer lifetime value based on transaction and demographic data. All of these transformations can be performed in Clementine, using a variety of Field Operations nodes.
Several nodes in Clementine provide the ability to derive new fields:

<table>
<thead>
<tr>
<th>Node name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derive</td>
<td>The Derive node is the most all-encompassing way to <em>manually</em> derive a variety of new set, flag, and range fields.</td>
</tr>
<tr>
<td>Reclassify</td>
<td>For existing sets, the Reclassify node helps you map values to new categories.</td>
</tr>
<tr>
<td>Binning</td>
<td>For numeric range fields, the Binning node <em>automatically</em> creates set fields based on p-tiles, mean/standard deviation, ranks, and fixed-width bins.</td>
</tr>
<tr>
<td>Set to Flag</td>
<td>The Set to Flag node is used to derive multiple flag fields based on the values for one or more set fields.</td>
</tr>
<tr>
<td>History</td>
<td>The History node enables you to create new fields containing data from fields in previous records (for example, sequential data, such as time series data).</td>
</tr>
</tbody>
</table>

**Using the Derive Node**

Using the Derive node, you can create six types of new fields from one or more existing fields:

- **Formula.** The new field is the result of an arbitrary CLEM expression.
- **Flag.** The new field is a flag, representing a specified condition.
- **Set.** The new field is a set, meaning that its members are a group of specified values.
- **State.** The new field is one of two states. Switching between these states is triggered by a specified condition.
- **Count.** The new field is based on the number of times that a condition has been true.
- **Conditional.** The new field is the value of one of two expressions, depending on the value of a condition.

Each of these nodes contains a set of special options in the Derive node dialog box. These options are discussed in subsequent topics.
**Setting Basic Options for the Derive Node**

At the top of the dialog box for Derive nodes are a number of options for selecting the type of Derive node that you need.

**Figure 7-14**
Derive node dialog box

**Mode.** Select Single or Multiple, depending on whether you want to derive multiple fields. When Multiple is selected, the dialog box changes to include options for multiple Derive fields.

**Derive field.** For simple Derive nodes, specify the name of the field that you want to derive and add to each record. The default name is DeriveN, where N is the number of Derive nodes that you have created thus far during the current session.

**Derive as.** Select a type of Derive node, such as Formula or Set, from the drop-down list. For each type, a new field is created based on the conditions that you specify in the type-specific dialog box.
Selecting an option from the drop-down list will add a new set of controls to the main dialog box according to the properties of each Derive node type.

**Field type.** Select a type, such as range, set, or flag, for the newly derived node. This option is common to all forms of Derive nodes.

*Note:* Deriving new fields often requires the use of special functions or mathematical expressions. To help you create these expressions, an Expression Builder is available from the dialog box for all types of Derive nodes and provides rule checking as well as a complete list of CLEM (Clementine Language for Expression Manipulation) expressions. For more information, see “What Is CLEM?” in Chapter 8 on page 205.

**Deriving Multiple Fields**

Setting the mode to Multiple within a Derive node gives you the capability to derive multiple fields based on the same condition within the same node. This feature saves time when you want to make identical transformations on several fields in your data set. For example, if you want to build a regression model predicting current salary based on beginning salary and previous experience, it might be beneficial to apply a log transformation to all three skewed variables. Rather than add a new Derive node for each transformation, you can apply the same function to all fields at once. Simply select all fields from which to derive a new field and then type the derive expression using the @FIELD function within the field parentheses.

*Note:* The @FIELD function is an important tool for deriving multiple fields at the same time. It allows you to refer to the contents of the current field or fields without specifying the exact field name. For instance, a CLEM expression used to apply a log transformation to multiple fields is \( \log(@FIELD) \).
The following options are added to the dialog box when you select Multiple mode:

**Derive from.** Use the Field Chooser to select fields from which to derive new fields. One output field will be generated for each selected field. *Note:* Selected fields do not need to be the same storage type; however, the Derive operation will fail if the condition is not valid for all fields.

**File name extension.** Type the extension that you would like added to the new field name(s). For example, for a new field containing the log of Current Salary, you could add the extension `log_` to the field name, producing `log_Current Salary`. Use the radio buttons to choose whether to add the extension as a prefix (at the beginning) or as a suffix (at the end) of the field name. The default name is DeriveN, where N is the number of Derive nodes that you have created thus far during the current session.
As in the single-mode Derive node, you now need to create an expression to use for deriving a new field. Depending on the type of Derive operation selected, there are a number of options to create a condition. These options are discussed in subsequent topics. To create an expression, you can simply type in the formula field(s) or use the Expression Builder by clicking the calculator button. Remember to use the @FIELD function when referring to manipulations on multiple fields.

**Selecting Multiple Fields**

For all nodes that perform operations on multiple input fields, such as Derive (multiple mode), Aggregate, Sort, and Multiplot, you can easily select multiple fields using the following dialog box.

**Figure 7-16**

*Selecting multiple fields*

**Sort by.** You can sort available fields for viewing by selecting one of the following options:

- **Natural.** View the order of fields as they have been passed down the data stream into the current node.
- **Name.** Use alphabetical order to sort fields for viewing.
- **Type.** View fields sorted by their type. This option is useful when selecting fields by type.
Select fields from the table one at a time or use the Shift-click and Ctrl-click methods to select multiple fields. You can also use the buttons below to select groups of fields based on their type or to select or deselect all fields in the table.

**Setting Derive Formula Options**

Derive Formula nodes create a new field for each record in a data set based on the results of a CLEM expression. Note that this expression cannot be conditional. To derive values based on a conditional expression, use the flag or conditional type of Derive node.

*Figure 7-17  Setting options for a Derive Formula node*

**Formula.** Specify a formula using the CLEM language to derive a value for the new field. For example, using the *P3_LoS* stream shipped with the Clementine Application Template (CAT) for CRM, you can derive the length of service for contracts...
pertaining to all customers in the database. The new field is called LoS and using the Expression Builder, you can create the following expression in the Formula field:

\[
\text{date\_years\_difference(\text{CardStart}\_\text{Date},'20010101')}
\]

Upon execution, the new LoS field will be created for each record and will contain the value of the difference between the value for \text{CardStart}\_\text{Date} and the reference date (2001/01/01) for each record.

### Setting Derive Flag Options

Derive Flag nodes are used to indicate a specific condition, such as high blood pressure or customer account inactivity. A flag field is created for each record, and when the true condition is met, the flag value for true is added in the field.

**Figure 7-18**

*Deriving a flag field to indicate inactive accounts*
**True value.** Specify a value to include in the flag field for records that match the condition specified below. The default is T.

**False value.** Specify a value to include in the flag field for records that do not match the condition specified below. The default is F.

**True when.** Specify a CLEM condition to evaluate certain values of each record and give the record a **True** value or a **False** value (defined above). Note that the true value will be given to records in the case of non-false numeric values.

*Note:* To return an empty string, you should type opening and closing quotes with nothing between them, such as “”. Empty strings are often used, for example, as the false value in order to enable true values to stand out more clearly in a table. Similarly, quotes should be used if you want a string value that would otherwise be treated as a number.

### Setting Derive Set Options

Derive Set nodes are used to execute a set of CLEM conditions in order to determine which condition each record satisfies. As a condition is met for each record, a value (indicating which set of conditions was met) will be added to the new, derived field.
**Default value.** Specify a value to be used in the new field if none of the conditions are met.

**Set field to.** Specify a value to enter in the new field when a particular condition is met. Each value in the list has an associated condition that you specify in the adjacent column.

**If this condition is true.** Specify a condition for each member in the set field to list. Use the Expression Builder to select from available functions and fields. You can use the arrow and delete buttons to reorder or remove conditions.

A condition works by testing the values of a particular field in the data set. As each condition is tested, the values specified above will be assigned to the new field to indicate which, if any, condition was met. If none of the conditions are met, the default value is used.
**Setting Derive State Options**

Derive State nodes are somewhat similar to Derive Flag nodes. A Flag node sets values depending on the fulfillment of a *single* condition for the current record, but a Derive State node can change the values of a field depending on how it fulfills *two independent* conditions. This means that the value will change (turn On or Off) as each condition is met.

**Figure 7-20**
*Using a Derive State node to indicate the current status of power plant conditions*

![](image)

**Initial state.** Select whether to give each record of the new field the On or Off value initially. Note that this value can change as each condition is met.

**“On” value.** Specify the value for the new field when the On condition is met.

**Switch “On” when.** Specify a CLEM condition that will change the state to On when the condition is true. Click the calculator button to open the Expression Builder.
“Off” value. Specify the value for the new field when the Off condition is met.

Switch “Off when. Specify a CLEM condition that will change the state to Off when the condition is false. Click the calculator button to open the Expression Builder.

Note: To specify an empty string, you should type opening and closing quotes with nothing between them, such as “ “. Similarly, quotes should be used if you want a string value that would otherwise be treated as a number.

Setting Derive Count Options

A Derive Count node is used to apply a series of conditions to the values of a numeric field in the data set. As each condition is met, the value of the derived count field is increased by a set increment. This type of Derive node is useful for time series data.

Figure 7-21
Count options in the Derive node dialog box
**Initial value.** Sets a value used on execution for the new field. The initial value must be a numeric constant. Use the arrow buttons to increase or decrease the value.

**Increment when.** Specify the CLEM condition that, when met, will change the derived value based on the number specified in Increment by. Click the calculator button to open the Expression Builder.

**Increment by.** Set the value used to increment the count. You can use either a numeric constant or the result of a CLEM expression.

**Reset when.** Specify a condition that, when met, will reset the derived value to the initial value. Click the calculator button to open the Expression Builder.

**Setting Derive Conditional Options**

Derive Conditional nodes use a series of If, Then, Else statements to derive the value of the new field.
If. Specify a CLEM condition that will be evaluated for each record upon execution. If the condition is true (or non-false, in the case of numbers), the new field is given the value specified below by the Then expression. Click the calculator button to open the Expression Builder.

Then. Specify a value or CLEM expression for the new field when the If statement above is true (or non-false). Click the calculator button to open the Expression Builder.

Else. Specify a value or CLEM expression for the new field when the If statement above is false. Click the calculator button to open the Expression Builder.
Filler Node

Filler nodes are used to replace field values and change storage. You can choose to replace values based on a specified CLEM condition, such as `@BLANK(@FIELD)`. Alternatively, you can choose to replace all blanks or null values with a specific value. Filler nodes are often used in conjunction with the Type node to replace missing values. For example, you can fill blanks with the mean value of a field by specifying an expression such as `@GLOBAL_MEAN`. This expression will fill all blanks with the mean value as calculated by a Set Globals node.

![Filler node dialog box](image)

**Fill in fields.** Using the Field Chooser (button to the right of the text field), select fields from the data set whose values will be examined and replaced. The default behavior is to replace values depending on the Condition and Replace with expressions specified below. You can also select an alternative method of replacement using the Replace options below.
Chapter 7

Note: When selecting multiple fields to replace with a user-defined value, it is important that the field types are similar (all numeric or all symbolic).

Replace. Select to replace the values of the selected field(s) using one of the following methods:

- **Based on condition.** This option activates the Condition field and Expression Builder for you to create an expression used as a condition for replacement with the value specified.

- **Always.** Replaces all values of the selected field. For example, you could use this option to convert the storage of income to a string using the following CLEM expression: `(to_string(income))`.

- **Blank values.** Replaces all user-specified blank values in the selected field. The standard condition `@BLANK(@FIELD)` is used to select blanks. Note: You can define blanks using the Types tab of the source node or with a Type node.

- **Null values.** Replaces all system null values in the selected field. The standard condition `@NULL(@FIELD)` is used to select nulls.

- **Blank and null values.** Replaces both blank values and system nulls in the selected field. This option is useful when you are unsure whether or not nulls have been defined as missing values.

Condition. This option is available when you have selected the Based on condition option. Use this text box to specify a CLEM expression for evaluating the selected fields. Click the calculator button to open the Expression Builder, an interactive way to build CLEM expressions.

Replace by. Specify a CLEM expression to give a new value to the selected fields. You can also replace the value with a null value by typing `undef` in the text box. Click the calculator button to open the Expression Builder, an interactive way to build CLEM expressions.

Note: When the field(s) selected are string, you should replace them with a string value. Using the default 0 or another numeric value as the replacement value for string fields will result in an error.
**Storage Conversion Using the Filler Node**

Using the Replace condition of a Filler node, you can easily convert the field storage for single or multiple fields. For example, using the conversion function `to_integer`, you could convert `income` from a string to an integer using the following CLEM expression: `to_integer(income)`.

**Figure 7-24**
*Using a Filler node to convert field storage*

The following conversion functions are available:
- `to_integer(ITEM)`
- `to_real(ITEM)`
- `to_string(ITEM)`
- `to_time(ITEM)`
- `to_timestamp(ITEM)`
You can view available conversion functions and automatically create a CLEM expression using the Expression Builder. From the Functions drop-down list, select Conversion to view a list of storage conversion functions.

**Reclassify Node**

The Reclassify node enables the transformation from one set of discrete values to another. Reclassification is useful for collapsing categories or regrouping data for analysis. For example, you could reclassify the values for Product name into three groups, such as Kitchenware; Clothing, Bath and Linens; and Appliances. Often, this operation is performed directly from a Distribution node by grouping values and generating a Reclassify node. For more information, see “Using a Distribution Graph” in Chapter 10 on page 261.

Reclassification can be performed for one or more symbolic fields. You can also choose to substitute the new values for the existing field or generate a new field.

Before using a Reclassify node, consider whether another Field Operations node is more appropriate for the task at hand:

- To transform numeric ranges into sets using an automatic method, such as ranks or percentiles, you should use a Binning node.
- To classify numeric ranges into sets manually, you should use a Derive node. For example, if you want to collapse salary values into specific salary range categories, you should use a Derive node to define each category manually.
- To create one or more flag fields based on the values of a categorical field, such as Mortgage_type, you should use a Set to Flag node.

**Setting Options for the Reclassify Node**

There are three steps to using the Reclassify node:

- First, select whether you want to reclassify multiple fields or a single field.
- Next, choose whether to recode into the existing field or create a new field.
- Then, use the dynamic options in the Reclassify node dialog box to map sets as desired.
Figure 7-25
Reclassify node dialog box

**Mode.** Select Single to reclassify the categories for one field. Select Multiple to activate options enabling the transformation of more than one field at a time.

**Reclassify into.** Select New field to keep the original set field and derive an additional field containing the reclassified values. Select Existing field to overwrite the values in the original field with the new classifications. This is essentially a “fill” operation.

Once you have specified mode and replacement options, you must select the transformation field and specify the new classification values using the dynamic options on the bottom half of the dialog box. These options vary depending on the mode you have selected above.
Reclassify field(s). Use the Field Chooser button on the right to select one (Single mode) or more (Multiple mode) discrete fields.

New field name. Specify a name for the new set field containing recoded values. This option is available only in Single mode when New field is selected above. When Existing field is selected, the original field name is retained. When working in Multiple mode, this option is replaced with controls for specifying an extension added to each new field. For more information, see “Reclassifying Multiple Fields” on page 187.

Reclassify values. This table enables a clear mapping from old set values to those you specify here.

- **Original value.** This column lists existing values for the select field(s).
- **New value.** Use this column to type new category values or select one from the drop-down list.

- Click Get to read original values for one or more fields selected above.
- Click Copy to paste original values over to the New value column for fields that have not been mapped yet. The unmapped original values are added to the drop-down list.
- Click Clear new to erase all specifications in the New value column. Note: This option does not erase the values from the drop-down list.
- Click Auto to automatically generate consecutive integers for each of the original values. Only integer values (no real values, such as 1.5, 2.5, etc.) can be generated.

Figure 7-26
Auto-classification dialog box

For example, you can automatically generate consecutive product ID numbers for product names or course numbers for university class offerings. This functionality corresponds to the Automatic Recode transformation for sets in SPSS.
For unspecified values use. This option is used for filling unspecified values in the new field. You can either choose to keep the original value by selecting Original value or specify a default value.

Reclassify Multiple Fields

To map category values for more than one field at a time, set the mode to Multiple. This enables new settings in the Reclassify dialog box, which are described below.

Figure 7-27
Dynamic dialog box options for reclassifying multiple fields
Reclassify fields. Use the Field Chooser button on the right to select the fields that you want to transform. Using the Field Chooser, you can select all fields at once or fields of a similar type, such as set or flag.

Field name extension. When recoding multiple fields simultaneously, it is more efficient to specify a common extension added to all new fields rather than individual field names. Specify an extension such as _recode and select whether to append or prepend this extension to the original field names.

Storage and Type for Reclassified Fields

The Reclassify node always creates a Set type field from the recode operation. In some cases, this may change the type of the field when using the Existing field reclassification mode.

The new field's storage (how data is stored rather than how it is used) is calculated based on the following Settings tab options:

- If unspecified values are set to use a default value, the storage type is determined by examining both the new values as well as the default value and determining the appropriate storage. For example, if all values can be parsed as integers, the field will have the integer storage type.
- If unspecified values are set to use the original values, the storage type is based on the storage of the original field. If all of the values can be parsed as the storage of the original field, then that storage is preserved; otherwise, the storage is determined by finding the most appropriate storage type encompassing both old and new values.

Note: If the original type was uninstantiated, the new type will be also be uninstantiated.

Binning Node

The Binning node enables you to automatically create new set fields based on the values of one or more existing numeric range fields. For example, you can transform a scale income field into a new categorical field containing groups of income as deviations from the mean. In SPSS, this is also known as Automatic Recode. Once
you have created bins for the new field, you can generate a Derive node based on the cut points.

Before using a Binning node, consider whether another Field Operations node is more appropriate for the task at hand:

- To manually specify cut points for categories, such as specific predefined salary ranges, use a Derive node.
- To create new categories for existing sets, use a Reclassify node.

**Missing Value Handling**

The Binning node handles missing values in the following ways:

- **User-specified blanks.** Missing values specified as blanks are included during the transformation. For example, if you designated –99 to indicate a blank value using the Type node, this value will be included in the binning process. To ignore blanks during binning, you should use a Filler node to replace the blank values with the system null value.

- **System-missing values ($null$).** Null values are ignored during the binning transformation and remain nulls after the transformation.

The Settings tab provides options for available techniques. The View tab displays cut points established for data previously run through the node.

**Setting Options for the Binning Node**

Using the Binning node, you can automatically generate bins (categories) using the following techniques:

- Fixed-width binning
- Tiles (equal counts)
- Mean and standard deviation
- Ranks

The bottom half of the dialog box changes dynamically depending on the binning method you select.
**Bin fields.** Numeric range fields pending transformation are displayed here. The Binning node enables you to bin multiple fields simultaneously. Add or remove fields using the buttons on the right.

**Binning method.** Select the method used to determine cut points for new field bins (categories).

The following topics discuss options for the available methods of binning.

**Fixed-Width Bins**

When you choose Fixed-width as the binning method, a new set of options is displayed in the dialog box.
**Name extension.** Specify an extension to use for the generated field(s). _BIN_ is the default extension. You may also specify whether the extension is added to the start (Prefix) or end (Suffix) of the field name. For example, you could generate a new field called income_BIN.

**Bin width.** Specify a value (integer or real) used to calculate the “width” of the bin. For example, you can use the default value, 10, to bin the field Age. Since Age has a range from 18–65, the generated bins would be the following:

**Table 7-1**

Bins for Age with range 18–65

<table>
<thead>
<tr>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
<th>Bin 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;=13 to &lt;23</td>
<td>&gt;=23 to &lt;33</td>
<td>&gt;=33 to &lt;43</td>
<td>&gt;=43 to &lt;53</td>
<td>&gt;=53 to &lt;63</td>
<td>&gt;=63 to &lt;73</td>
</tr>
</tbody>
</table>

The start of bin intervals is calculated using the lowest scanned value minus half the bin width (as specified). For example, in the bins shown above, 13 is used to start the intervals according to the following calculation: 18 \[ \text{lowest data value} \] − 5 \[ 0.5 \times (\text{Bin width of 10}) \] = 13.

**No. of bins.** Use this option to specify an integer used to determine the number of fixed-width bins (categories) for the new field(s).
Once you have executed the Binning node in a stream, you can view the bin thresholds generated by clicking the Generate tab in the Binning node dialog box. For more information, see “Viewing Generated Bins” on page 196.

Tiles (Equal Count)

Equal count, or equal frequency, bins are generated by splitting scanned records into percentile groups containing the same number of cases. Values are assigned based on membership in a particular percentile. For example, quartiles would assign a rank of 1 to cases below the 25th percentile, 2 to cases between the 25th and 50th percentiles, 3 to cases between the 50th and 75th percentiles, and 4 to cases above the 75th percentile.

When you choose Tiles (equal count) as the binning method, a new set of options is displayed in the dialog box.

Figure 7-30
Binning node dialog box: Settings tab with options for equal count bins

<table>
<thead>
<tr>
<th>Equal Count Binning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tile name extension: TILE</td>
</tr>
<tr>
<td>Custom tile extension: TILEN</td>
</tr>
<tr>
<td>□ Quartile (4)</td>
</tr>
<tr>
<td>□ Vingtile (20)</td>
</tr>
<tr>
<td>□ Custom N</td>
</tr>
<tr>
<td>Ties: ○ Add to next ○ Keep in current</td>
</tr>
</tbody>
</table>

**Tile name extension.** Specify an extension used for field(s) generated using standard p-tiles. The default extension is _TILE_ plus N, where N is the tile number. You may also specify whether the extension is added to the start (Prefix) or end (Suffix) of the field name. For example, you could generate a new field called income_BIN4.

**Custom tile extension.** Specify an extension used for custom percentiles. The default is _TILEN_. N in this case will not be replaced by the custom number.

Available p-tiles are:
- **Quartile.** Select to generate four percentile bins, each containing 25% of the cases.
- **Quintile.** Select to generate five percentile bins, each containing 20% of the cases.
- **Decile.** Select to generate 10 percentile bins, each containing 10% of the cases.
- **Vingtile.** Select to generate 20 percentile bins, each containing 5% of the cases.
- **Percentile.** Select to generate 100 percentile bins, each containing 1% of the cases.
- **Custom N.** Select to specify the width of each interval, expressed as a percentage of the total number of cases. For example, a value of 33.3 would produce three banded categories (two cut points), each containing 33.3% of the cases.

*Note:* Bin IDs (values) are assigned sequentially. This means that where there are fewer discrete values than tiles specified, all tiles will not be used. In such cases, the new distribution is likely to reflect the original distribution of your data.

**Ties.** When values on either side of a percentile cut point (that is, 25% of cases) are identical, this results in a “tie” condition. You can handle the tie in two ways:

- **Add to next.** Select to move the tie values up to the next bin, making that bin larger than its specified percentile.
- **Keep in current.** Select to move tie values lower, keeping them in the current bin.

Depending on which option you select, values may be assigned differently for the same set of numbers. For example, the table below illustrates how simplified field values are recoded as quartiles depending on the selected ties option.

**Table 7-2**
*Comparison of bin IDs by ties option*

<table>
<thead>
<tr>
<th>Values</th>
<th>Add to Next</th>
<th>Keep in Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

The number of items per bin is calculated as:

\[
\text{total number of value} / \text{number of tiles}
\]
In the simplified example above, the desired number of items per bin is 1.25 (5 values / 4 quartiles). The value 13 (being value number 2) straddles the 1.25 desired count threshold and is therefore treated differently depending on the selected ties option. In Add to Next mode, it is added into bin 2. In Keep in Current mode, it is left in bin 1, pushing the range of values for bin 4 outside that of existing data values.

**Rank Cases**

When you choose Ranks as the binning method, a new set of options is displayed in the dialog box.

![Binning node dialog box: Settings tab with options for ranks](image)

Ranking creates new fields containing ranks, fractional ranks, and percentile values for numeric fields depending on the options specified below.

**Rank order.** Select Ascending (lowest value is marked 1) or Descending (highest value is marked 1).

**Rank.** Select to rank cases in ascending or descending order as specified above. The range of values in the new field will be 1–N, where N is the number of discrete values in the original field. Tied values are given the average of their rank.

**Fractional rank.** Select to rank cases where the value of the new field equals rank divided by the sum of the weights of the nonmissing cases. Fractional ranks fall in the range of 0–1.
Percentage fractional rank. Each rank is divided by the number of records with valid values and multiplied by 100. Percentage fractional ranks fall in the range of 1–100.

Extension. For all rank options, you can create custom extensions and specify whether the extension is added to the start (Prefix) or end (Suffix) of the field name. For example, you could generate a new field called income_P_RANK.

**Mean/Standard Deviation**

When you choose Mean/standard deviation as the binning method, a new set of options is displayed in the dialog box.

*Figure 7-32*

*Binning node dialog box: Settings tab with options for mean/standard deviation*

This method generates one or more new fields with banded categories based on the values of the mean and standard deviation of the distribution of the specified field(s). Select the number of deviations to use below.

Name extension. Specify an extension to use for the generated field(s). _SDBIN is the default extension. You may also specify whether the extension is added to the start (Prefix) or end (Suffix) of the field name. For example, you could generate a new field called income_SDBIN.

- +/- 1 standard deviation. Select to generate three bins
- +/- 2 standard deviations. Select to generate five bins.
- +/- 3 standard deviations. Select to generate seven bins.
For example, selecting +/-1 standard deviation results in the three bins as calculated below:

<table>
<thead>
<tr>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x &lt; (\text{Mean} - \text{Std. Dev})$</td>
<td>$(\text{Mean} - \text{Std. Dev}) \leq x \leq (\text{Mean} + \text{Std. Dev})$</td>
<td>$x &gt; (\text{Mean} + \text{Std. Dev})$</td>
</tr>
</tbody>
</table>

In a normal distribution, 68% of the cases fall within 1 standard deviation of the mean, 95% within 2 standard deviations, and 99% within 3 standard deviations.

**Viewing Generated Bins**

Using the Generate tab for the Binning node, you can view the thresholds for each of the input fields that have been categorized.

**Figure 7-33**
*Binning node dialog box: Generate tab*

**Binned field.** Use the drop-down list to select a field for viewing. Field names shown use the original field name for clarity.
**Tile.** Use the drop-down list to select a tile, such as 10 or 100, for viewing. This option is available only when bins have been generated using the equal counts method.

**Bin Thresholds.** Threshold values are shown here for each generated bin. You cannot change the values shown here, but you can click the Generate Derive button to create a Derive node with the current values in an editable form.

- Thresholds are available only after you have run data through the node (that is, “executed” the stream).
- Thresholds will be overwritten when new data is run through the stream.

**Generate Derive.** Generating a Derive node is useful for applying established bin thresholds from one set of data to another. Furthermore, once these split points are known, a Derive operation is more efficient than a Binning operation when working with large datasets.

**Set to Flag Node**

The Set to Flag node is used to derive multiple flag fields based on the symbolic values defined for one or more set fields. For example, you may have purchased data on several products that can be bought in several different departments within a store. Currently, your data consist of one product per purchase and lists the product code and the department code (a set) as two attributes. For easier data manipulation, you can create a flag field for each department, which will indicate whether or not the product was purchased in that department.
Figure 7-34
Creating a flag field for high blood pressure using the drug demo data

Setting Options for the Set to Flag Node

Set to Flag nodes require fully instantiated types (data whose type attributes and values are known). Therefore, you must have a Type node upstream from a Set to Flag node that specifies one or more fields as set types. After all options have been specified in the Set to Flag node, you can detach the Type node, if desired, since it is no longer needed for information on types.

Set fields. Lists all fields in the data whose types are set. Select one from the list to display the values in the set. You can choose from these values to create a flag field.
Field name extension. Select to enable controls for specifying an extension that will be added as a suffix or prefix to the new flag field. By default, new field names are automatically created by combining the original field name with the field value into a label, such as Fieldname_fieldvalue.

Available set values. Values in the set selected above are displayed here. Select one or more values for which you want to generate flags. For example, if the values in a field called blood_pressure are High, Medium, and Low, you can select High and add it to the list on the right. This will create a field with a flag for records with a value indicating high blood pressure.

Create flag fields. The newly created flag fields are listed here. You can specify options for naming the new field using the field name extension controls.

True value. Specify the true value used by the node when setting a flag. By default, this value is T.

False value. Specify the false value used by the node when setting a flag. By default, this value is F.

Aggregate keys. Select to group records together based on key fields specified below. When Aggregate keys is selected, all flag fields in a group will be “turned on” if any record was set to true. Use the Field Chooser to specify which key fields will be used to aggregate records.

History Node

History nodes are most often used for sequential data, such as time series data. They are used to create new fields containing data from fields in previous records. When using a History node, you may want to have data that is presorted by a particular field. You can use a Sort node to do this.
**Setting Options for the History Node**

Figure 7-35
*History node dialog box*

**Selected fields.** Using the Field Chooser (button to the right of the text box), select the fields for which you want a history. Each selected field is used to create new fields for all records in the data set.

**Offset.** Specify the latest record prior to the current record from which you want to extract historical field values. For example, if Offset is set to 3, as each record passes through this node, the field values for the third record previous will be included in the current record. Use the Span settings to specify how far back records will be extracted from. Use the arrows to adjust the offset value.

**Span.** Specify how many prior records from which you want to extract values. For example, if Offset is set to 3 and Span is set to 5, each record that passes through the node will have five fields added to it for each field specified in the Selected fields list. This means that when the node is processing record 10, fields will be added from record 7 through record 3. Use the arrows to adjust the span value.
**Where history is unavailable.** Select one of the following three options for handling records that have no history values. This usually refers to the first several records at the top of the data set, for which there are no previous records to use as a history.

- **Discard records.** Select to discard records where no history value is available for the field selected.
- **Leave history undefined.** Select to keep records where no history value is available. The history field will be filled with an undefined value, displayed as $null$.
- **Fill values with.** Specify a value or string to be used for records where no history value is available. The default replacement value is *undef*, the system null. Null values are displayed in Clementine using the string $null$. When selecting a replacement value, keep in mind the following rules in order for proper execution to occur:
  - Selected fields should be of the same storage type.
  - If all the selected fields have numeric storage, the replacement value must be parsed as an integer.
  - If all the selected fields have real storage, the replacement value must be parsed as a real.
  - If all the selected fields have symbolic storage, the replacement value must be parsed as a string.
  - If all the selected fields have date/time storage, the replacement value must be parsed as a date/time field.

If none of the above conditions are met, you will receive an error when executing the History node.

**Field Reorder Node**

The Field Reorder node enables you to define the natural order used to display fields downstream. This order affects the display of fields in a variety of places, such as tables, lists, and the Field Chooser. This operation is useful, for example, when working with wide data sets to make fields of interest more visible.
Chapter 7

Setting Field Reorder Options

There are two ways to reorder fields: **custom ordering** and **automatic sorting**.

**Custom Ordering**

Select Custom Order to enable a table of field names and types where you can view all fields and use arrow buttons to create a custom order.

Figure 7-36
Reordering to display fields of interest first

To reorder fields:

- Select a field in the table. Use the Ctrl-click method to select multiple fields.
- Use the simple arrow buttons to move the field(s) up or down one row.
- Use the line-arrow buttons to move the field(s) to the bottom or top of the list.
Specify the order of fields not included here by moving up or down the divider row, indicated as [other fields].

**Other fields.** The purpose of the [other fields] divider row is to break the table into two halves.

- Fields appearing above the divider row will be ordered (as they appear in the table) at the top of all natural orders used to display the fields downstream of this node.
- Fields appearing below the divider row will be ordered (as they appear in the table) at the bottom of all natural orders used to display the fields downstream of this node.

**Figure 7-37**
*Diagram illustrating how “other fields” are incorporated into the new field order.*

<table>
<thead>
<tr>
<th>Field Reorder Table</th>
<th>New Natural Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>dep</td>
<td>dep</td>
</tr>
<tr>
<td>realincome</td>
<td>realincome</td>
</tr>
<tr>
<td>[other fields]</td>
<td>children</td>
</tr>
<tr>
<td>age</td>
<td>marital</td>
</tr>
<tr>
<td>sex</td>
<td>mortgage</td>
</tr>
<tr>
<td>region</td>
<td>age</td>
</tr>
<tr>
<td></td>
<td>sex</td>
</tr>
<tr>
<td></td>
<td>region</td>
</tr>
</tbody>
</table>

- All other fields not appearing in the field reorder table will appear between these “top” and “bottom” fields as indicated by the placement of the divider row.

Additional custom sorting options include:

- Sort fields in ascending or descending order by clicking on the arrows above each column header (Type, Name, and Storage). When sorting by column, fields not specified here (indicated by the [other fields] row) are sorted last in their natural order.
- Click Clear Unused to delete all unused fields from the Field Reorder node. Unused fields are displayed in the table with a red font. This indicates that the field has been deleted in upstream operations.
- Specify ordering for any new fields (displayed with a lightning icon to indicate a new or unspecified field). When you click OK or Apply, the icon disappears.

**Note:** If fields are added upstream after a custom order has been applied, the new fields will be appended at the bottom of the custom list.
Chapter 7

**Automatic Sorting**

Select Automatic Sort to specify a parameter for sorting. The dialog box options dynamically change to provide options for automatic sorting.

![Figure 7-38](image)

*Reordering all fields using automatic sorting options*

- **Sort By.** Select one of three ways to sort fields read into the Reorder node. The arrow buttons indicate whether the order will be ascending or descending. Select one to make a change.
  - Name
  - Type
  - Storage

Fields added upstream of the Field Reorder node after auto-sort has been applied will automatically be placed in their proper position based on the sort type selected.
Building CLEM Expressions

What Is CLEM?

The Clementine Language for Expression Manipulation (CLEM) is a powerful language for analyzing and manipulating the data that flows along Clementine streams. Data miners use CLEM extensively in stream operations to perform tasks as simple as deriving profit from cost and revenue data or as complex as transforming Web-log data into a set of fields and records with usable information.

CLEM is used within Clementine to:
- Compare and evaluate conditions on record fields.
- Derive values for new fields.
- Derive new values for existing fields.
- Reason about the sequence of records.
- Insert data from records into reports.

A subset of the CLEM language can be used when scripting either in the user interface or batch mode. This allows you to perform many of the same data manipulations in an automated fashion. For more information, see “Introduction to Scripting” in Chapter 18 on page 597.
Values and Data Types

CLEM expressions are similar to formulas constructed from values, field names, operators, and functions. The simplest valid CLEM expression is a value or a field name. Examples of valid values are:

3
1.79
'banana'

Examples of field names are:

Product_ID
'$P\text{-NextField}''

Where Product is the name of a field from a market basket data set, '$P\text{-NextField}'' is the name of a parameter, and the value of the expression is the value of the named field. Typically, field names start with a letter and may also contain digits and underscores (_). You can use names that do not follow these rules if you place the name within quotation marks. CLEM values can be any of the following:

- Strings—for example, "c1", "Type 2", "a piece of free text"
- Integers—for example, 12, 0, −189
- Real numbers—for example, 12.34, 0.0, −0.0045
- Date/time fields—for example, 05/12/2002, 12/05/2002, 12/05/02

It is also possible to use the following elements:

- Character codes—for example, ‘a’ or 3
- Lists of items—for example, [1 2 3], ['Type 1' 'Type 2']

Character codes and lists do not usually occur as field values. Typically, they are used as arguments of CLEM functions.

Quoting Rules

Although the software is flexible when determining the fields, values, parameters, and strings used in a CLEM expression, the following general rules provide a list of “best practices” to use when creating expressions.
Strings—Always use double quotes when writing strings ("Type 2" or “value”). Single quotes may be used instead but at the risk of confusion with quoted fields.

Characters—Always use single back-slash quotes like this \ . For example, note the character d in the following function: stripchar( ‘d ‘,”drugA”). The only exception to this is when using an integer to refer to a specific character in a string. For example, note the character 5 in the following function: lowertoupper(“druga”(5)) —> “A”. Note: On a standard UK and US keyboard, the key for the backquote character (grave accent, unicode 0060) can be found just below the Escape key.

Fields—Fields are typically unquoted when used in CLEM expressions (subscr(2,arrayID)) —> CHAR). You may use single quotes when necessary to enclose spaces or other special characters (‘Order Number’). Fields that are quoted but undefined in the data set will be misread as strings.

Parameters—Always use single quotes ('$P-threshold').

Expressions and Conditions

CLEM expressions can return a result (used when deriving new values)—for example:

Weight * 2.2
Age + 1
sqrt(Signal-Echo)

Or, they can evaluate true or false (used when selecting on a condition)—for example:

Drug = "drugA"
Age < 16
not(PowerFlux) and Power > 2000

You can combine operators and functions arbitrarily in CLEM expressions—for example:

sqrt(abs(Signal)) * max(T1, T2) + Baseline

Brackets and operator precedence determine the order in which the expression is evaluated. In this example, the order of evaluation is:

abs(Signal) is evaluated, and sqrt is applied to its result.
max(T1, T2) is evaluated.
The two results are multiplied: x has higher precedence than +.
Finally, Baseline is added to the result.

The descending order of precedence (that is, operations that are executed first to operations that are executed last) is as follows:

- Function arguments
- Function calls
- xx
- x / mod div rem
- + -
- > < >= <= /= == == /=

If you want to override precedence, or if you are in any doubt of the order of evaluation, you can use parentheses to make it explicit—for example,

sqrt(abs(Signal)) * (max(T1, T2) + Baseline)

**CLEM Examples**

To illustrate correct syntax as well as the types of expressions possible with CLEM, example expressions follow.

**Simple Expressions**

Formulas can be as simple as this one, which derives a new field based on the values of fields After and Before:

(After - Before) / Before * 100.0

Notice that field names are unquoted when referring to the values of the field.
Similarly, the following expression simply returns the log of each value for the field salary.

log(salary)
Complex Expressions

Expressions can also be lengthy and more complex. The following expression returns True if the value of two fields (\$KX-Kohonen and \$KY-Kohonen) fall within the specified ranges. Notice that here the field names are single quoted because the field names contain special characters:

\[
('\$KX-Kohonen' \geq -0.2635771036148072 \text{ and } '\$KX-Kohonen' \leq 0.3146203637123107 \\
\text{and } '\$KY-Kohonen' \geq -0.18975617885589602 \text{ and } \\
'\$KY-Kohonen' \leq 0.17674794197082522) \rightarrow T
\]

Several functions, such as string functions, require you to enter several parameters using correct syntax. For example, the function \texttt{subscrs} is used below to return the first character of a \texttt{produce_ID} field, indicating whether an item is organic, genetically modified, or conventional. The results of an expression are described by “\rightarrow Result”:

\[
\text{subscrs(1,produce_ID)} \rightarrow 'c' \text{ } \\
\text{Similarly, the following expression is } \\
\text{stripchar('3','123')} \rightarrow '12'
\]

It is important to note that characters are always encapsulated within single backquotes.

Combining Functions in an Expression

Frequently CLEM expressions consist of a combination of functions. The function below combines \texttt{subscr} and \texttt{lowertoupper} to return the first character of \texttt{produce_ID} and convert it to upper case.

\[
\text{lowertoupper(subscr(1,produce_ID)) } \rightarrow 'C' \\
\text{This same expression can be written in shorthand as: } \\
\text{lowertoupper(produce_ID(1)) } \rightarrow 'C' \text{ } \\
\]

Another commonly used combination of functions is shown below.
locchar_back( 'n ', (length(web_page)), web_page)

This expression locates the character 'n' within the values of field web_page reading backwards from the last character of the field value. By including the length function as well, the expression dynamically calculates the length of the current value rather than using a static number such as 7, which will be invalid for values with less than seven characters.

**Special Functions**

Numerous special functions (preceded with an @ symbol) are available. Commonly used functions include:

@BLANK('referrer ID') -> T

Frequently special functions are used in conjunction as illustrated in the following example—a commonly used method of flagging blanks in more than one field at a time:

@BLANK(@FIELD) -> T

Additional examples are discussed throughout the CLEM documentation. For more information, see “CLEM Reference Overview” in Appendix A on page 663.

**Using the Expression Builder**

This release of Clementine enables you to build CLEM expressions with ease. Using the Expression Builder (E-Builder), you can quickly build expressions for use in Clementine nodes without memorizing exact field names or the CLEM language. The E-Builder contains a complete list of CLEM functions and operators as well as data fields from the current stream. If data types are known, or instantiated, you can view even more information about fields using options in the Expression Builder dialog box.
The Expression Builder is available wherever you need to write a CLEM expression, including expressions for the Select, Balance, Derive, Filler, Plot, Multiplot, Analysis, Report, and Table nodes. The basic method for creating an expression is:

- Double-click functions and fields to add them to the expression window.
- Use the operand, or calculator, buttons to create an expression.
- Click Check to validate the current expression before closing the E-Builder.

*Note:* The Expression Builder is not supported in scripting or parameter settings.

### Accessing the Expression Builder

The Expression Builder is available from numerous dialog boxes that use CLEM expressions, such as those used for the Derive node.
212

Chapter 8

To access the Expression Builder:

- Click the calculator button on the right side of the dialog box.

Figure 8-2
Calculator button used to access the Expression Builder

Creating Expressions

The Expression Builder not only provides complete lists of fields, functions, and operators, it also provides access to data values if your data are instantiated.

To create an expression using the Expression Builder:

- Type in the expression window, using the function and field lists as references.

  or

- Select the desired fields and functions from the scrolling lists.

- Double-click or click the yellow arrow button to add the field or function to the expression window.

- Use the operand buttons in the center of the dialog box to insert the operations into the expression.

Selecting Functions

The function list displays all available CLEM functions and operators. Scroll to select a function from the list, or for easier searching, use the drop-down list to display a subset of functions or operators.
Available functions are grouped into categories for easier searching. Note that there are two categories that you may find particularly useful:

- **All Functions** contains a complete list of available CLEM functions.
- **Recently Used** contains a list of CLEM functions used within the current session.

After you have selected a group of functions, double-click to insert the function in the expression window at the point indicated by the position of the cursor.

### Selecting Fields, Parameters, and Global Variables

The field list displays all fields available at this point in the data stream. Scroll to select a field from the list. Double-click or use the yellow arrow key to add a field to the expression above. You can also use the Fields drop-down list to display available parameters and global variables.
Values for set and flag fields may be viewed from a variety of Clementine dialog boxes, such as the Expression Builder and Data Audit node output.

**Viewing or Selecting Values**

Values for set and flag fields may be viewed from a variety of Clementine dialog boxes, such as the Expression Builder and Data Audit node output.
**Selecting Values for the Expression Builder**

If the data are fully instantiated, meaning that storage, types, and values are known, you can also use this dialog box to add values to an expression in the Expression Builder.

**To select and add a value:**

- Select a field from the Fields list.
- Click the Value picker button to open a dialog box listing values for the selected field.

![Value picker button](image)

- Select a value from the list.
- Click Insert to add it to the CLEM expression at the cursor position.

**Checking CLEM Expressions**

Before closing the Expression Builder, take a moment to check the function that you created. Unchecked expressions are displayed in red. Click Check to validate the expression, checking the following:

- Proper quoting of values and field names
- Proper usage of parameters and global variables
- Valid usage of operators
- Existence of referenced fields
- Existence and definition of referenced globals

If errors are found, an alert is raised, and the offending string is highlighted in the expression window.
Also, if you manually created the expression by typing in the window, try creating the expression again using the lists and operator buttons. This method automatically adds the proper quotes for fields and values.

**Types of CLEM Functions**

The following types of CLEM functions are available when working with data in Clementine. You can enter these functions as code in a variety of dialog boxes, such as those for Derive and Set to Flag nodes, or you can use the Expression Builder to create valid CLEM expressions without memorizing function lists or field names.

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information</td>
<td>Used to gain insight into field values. For example, the function <code>is_string</code> returns a true flag for all records whose type is a string.</td>
</tr>
<tr>
<td>Conversion</td>
<td>Used to construct new fields or convert storage type. For example, the function <code>to_timestamp</code> converts the selected field to a timestamp.</td>
</tr>
<tr>
<td>Function Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Comparison</td>
<td>Used to compare field values to each other or to a specified string. For example, &lt;= is used to compare whether the values of two fields are lesser or equal.</td>
</tr>
<tr>
<td>Logical</td>
<td>Used to perform logical operations, such as if, then, else operations.</td>
</tr>
<tr>
<td>Numeric</td>
<td>Used to perform numeric calculations, such as the natural log of field values.</td>
</tr>
<tr>
<td>Trigonometric</td>
<td>Used to perform trigonometric calculations, such as the arccosine of a specified angle.</td>
</tr>
<tr>
<td>Bitwise</td>
<td>Used to manipulate integers as bit patterns.</td>
</tr>
<tr>
<td>Random</td>
<td>Used to randomly select items or generate numbers.</td>
</tr>
<tr>
<td>String</td>
<td>Used to perform a wide variety of operations on strings, such as stripchar, which allows you to remove a specified character.</td>
</tr>
<tr>
<td>Date and Time</td>
<td>Used to perform a variety of operations on date/time fields.</td>
</tr>
<tr>
<td>Sequence</td>
<td>Used to gain insight into the record sequence of a data set or perform operations based on that sequence.</td>
</tr>
<tr>
<td>Global</td>
<td>Used to access global values created by a Set Globals node. For example, @MEAN is used to refer to the mean average of all values for a field across the entire data set.</td>
</tr>
<tr>
<td>Blanks and Null</td>
<td>Used to access, flag, and frequently to fill user-specified blanks or system-missing values. For example, @BLANK(FIELD) is used to raise a true flag for records where blanks are present.</td>
</tr>
<tr>
<td>Special Fields</td>
<td>Used to denote the specific fields under examination. For example, @FIELD is used when deriving multiple fields.</td>
</tr>
</tbody>
</table>

**Common Uses of CLEM**

There are a number of operations possible using CLEM. See the following topics for a general introduction to the most common operations.

**Working with Strings**

There are a number of operations available for strings, including:

- Converting a string to uppercase or lowercase—uppertolower(CHAR).
- Removing specified characters, such as 'ID_' or '$' from a string variable—stripchar(CHAR,STRING).
Determining the length (number of characters) for a string variable—\texttt{length(STRING)}.

Checking the alphabetical ordering of string values—\texttt{alphabefore(STRING1, STRING2)}.

For more information, see “String Functions” in Appendix A on page 679.

**Handling Blanks and Missing Values**

Replacing blanks or missing values is a common data preparation task for data miners. CLEM provides you with a number of tools to automate blank handling. The Filler node is the most common place to work with blanks; however, the following functions can be used in any node that accepts CLEM expressions:

- \texttt{@BLANK(FIELD)} can be used to determine records whose values are blank for a particular field, such as \texttt{Age}.

- \texttt{@NULL(FIELD)} can be used to determine records whose values are system-missing for the specified field(s). In Clementine, system-missing values are displayed as \$null$ values.
Building CLEM Expressions

Figure 8-8
Using @NULL to fill missing values in the selected fields with 0

For more information, see “Functions Handling Blanks and Null Values” in Appendix A on page 695.

**Working with Numbers**

Numerous standard operations on numeric values are available in Clementine, such as:

- Calculating the sine of the specified angle—\( \sin(\text{NUM}) \)
- Calculating the natural log of numeric fields—\( \log(\text{NUM}) \)
- Calculating the sum of two numbers—\( \text{NUM1} + \text{NUM2} \)

For more information, see “Numeric Functions” in Appendix A on page 674.
Chapter 8

Working with Times and Dates

Time and date formats may vary depending on your data source and locale. The formats of date and time are specific to each stream and are set in the stream properties dialog box. The following examples are commonly used functions for working with date/time fields.

Calculating Time Passed

You can easily calculate the time passed from a baseline date using a family of functions similar to the one below. This function returns the time in months from the baseline date to the date represented by date string DATE, as a real number. This is an approximate figure, based on a month of 30.0 days.

\[ \text{date\_in\_months(} \text{Date} \text{)} \]

Comparing Date/Time Values

Values of date/time fields can be compared across records using functions similar to the one below. This function returns a value of true if date string \text{DATE1} represents a date prior to that represented by date string \text{DATE2}. Otherwise, this function returns a value of 0.

\[ \text{date\_before(} \text{Date1, Date2} \text{)} \]

Calculating Differences

You can also calculate the difference between two times and two dates using functions, such as:

\[ \text{date\_weeks\_difference(} \text{Date1, Date2} \text{)} \]

This function returns the time in weeks from the date represented by the date string \text{DATE1} to the date represented by date string \text{DATE2}, as a real number. This is based on a week of 7.0 days. If \text{DATE2} is prior to \text{DATE1}, this function returns a negative number.
Today's Date

The current date can be added to the data set using the function @TODAY. Today's date is added as a string to the specified field or new field using the date format selected in the stream properties dialog box. For more information, see “Date and Time Functions” in Appendix A on page 684.
Handling Missing Values

Overview of Missing Values

During the Data Preparation phase of data mining, you will often want to replace missing values in the data. **Missing values** are values in the data set that are unknown, uncollected, or incorrectly entered. Usually such values are invalid for their fields. For example, a field such as *Sex* should contain values such as *M* and *F*. If you discover the values *Y* or *Z* in the field, you can safely assume that such values are invalid and should therefore be interpreted as blanks. Likewise, a negative value for the field *Age* is meaningless and should also be interpreted as a blank. Frequently, such obviously wrong values are purposely entered or left blank during a questionnaire to indicate a nonresponse. At times you may want to examine these blanks more closely to determine whether a nonresponse, such as the refusal to give one's age, is a factor in predicting a specific outcome.

Some modeling techniques handle missing data better than others. For example, GRI, C5.0, and Apriori cope well with values that are explicitly declared as “missing” in a Type node. Other modeling techniques have trouble dealing with missing values and experience longer training times and result in less accurate models.

There are two types of missing values in Clementine:

- **System-missing values.** Also called **nulls**, these are values left blank in the database, and they have not been specifically set as “missing” in the Type node. System-missing values are displayed as $null$ in Clementine.

- **User-defined missing values.** Also called **blanks**, these are values, such as “unknown,” 99, or −1, that are explicitly defined in the Type node as missing. Data values specified as blanks are flagged for special treatment and are excluded from most calculations.
Chapter 9

Specifying Missing Values

Missing values in Clementine can be specified using the Type node, or the Types tab of a source node. In both locations, you can specify particular values as missing and decide whether to treat system nulls and white space as blanks. In the Missing column, select Specify from the drop-down list to open a dialog box in which you can specify options for missing values.

Explicitly setting blanks provides greater flexibility when treating missing values. For example, specifying system nulls as blank allows you to use the @BLANK function to treat these values along with other types of missing values.
Treating Missing Values

You should decide how to treat missing values in light of your business or domain knowledge. In order to ease training time and increase accuracy, you may want to remove blanks from your data set. On the other hand, the presence of blank values may lead to new business opportunities or additional insight.

There are several techniques used in Clementine for eliminating missing values. You can determine the best technique by addressing the following characteristics of your data:

- Size of the data set
- Number of fields containing blanks
- Amount of missing information

Once you have analyzed these factors, there are a couple of ways to treat missing values. These options revolve around removing fields and records or finding an adequate method of imputing values:

- Omitting the fields with missing values
- Omitting the records with missing values
- Filling in missing values with default values
- Filling in missing values with a value derived from a model

In determining which method to use, you should also consider the type of the field with missing values.

**Range types.** For numeric field types, such as range, you should always eliminate any non-numeric values before building a model because many models will not function if blanks are included in numeric fields.

**Discrete types.** For symbolic field types, such as set and flag, altering missing values is not necessary but will increase the accuracy of the model. For example, a model that uses the field Sex will still function with meaningless values, such as Y and Z, but removing all values other than M and F will increase the accuracy of the model.

Before making any final decisions, you can generate a data quality report of missing data using a Quality node. Once you have examined the report, you can use this node to automate the selection and filtering of records and fields with missing values.
You may encounter data sets in which the majority of missing values is concentrated in a small number of records. For example, a bank usually keeps detailed and complete records on its loan customers. If, however, the bank is less restrictive in approving loans for its own staff members, data gathered for staff loans is likely to have several blank fields. In such a case, there are two options for handling these missing values:

- You can use a Select node to remove the staff records.
- If the data set is large, you can use the @BLANK or @NULL functions in a Select node to discard all records with blanks. Note that when you are using @BLANK, it is helpful to use a Type node to specify blanks beforehand.

### Treating Fields with Missing Values

In some cases, it is advantageous to treat missing values at the field level rather than the record level. This approach allows you to experiment with the relative importance of particular fields before deciding on an approach for handing missing values.
values. Typically, the method used is based on the number of missing values in a particular attribute as well as the attribute's importance.

**Fields with Many Missing Values**

In some data sets, the majority of missing values is concentrated in a small number of fields. For example, a market research company may collect data from a general questionnaire containing 50 questions. Two of the questions address age and political persuasion, information that many people are reluctant to give. In this case, *Age* and *Political_persuasion* have many missing values. To handle these types of fields with many missing values, you have several options:

- You can use a Filter node to filter out the fields determined to have numerous missing values.
- Instead of removing the fields, you can use a Type node to set the fields' direction to None. This will keep the fields in the data set but leave them out of modeling processes.
- You can also choose to keep the fields and fill in missing values with sensible defaults, such as mean globals. This option is discussed further in the next topic.

**Fields with a Few Missing Values**

In many data sets, omissions and mistakes are made during data capture and data entry. For example, if inexperienced staff are processing numerous orders each day and entering the information into databases, the data set may contain some errant or missing values. In cases where there are only a few missing values, it is useful to insert values to replace the blanks. There are four methods commonly used for determining the replacement value.

- You can use a Type node to ensure that the field types cover only legal values and then set the *Check* column to Coerce for the fields whose blank values need replacing. For more information, see “Type Node” in Chapter 7 on page 148.
- You can use a Filler node to select the fields with missing values based on a specific condition. You can set the condition to test for those values and replace them using a specific value or a global variable created by the Set Globals node. For more information, see “Filler Node” in Chapter 7 on page 181.
Using both Type and Filler nodes, you can define blanks and replace them. First, use a Type node to specify information on what constitutes a missing value. Then use a Filler node to select fields whose values need replacing. For example, if the field Age is a range between 18 and 65 but also includes some spaces and negative values, select the White space option in the Specify Values dialog box of the Type node and add the negative values to the list of missing values. In the Filler node, select the field Age, set the condition to @BLANK(@FIELD), and change the Replace with expression to –1 (or some other numeric value).

The most ambitious option is to learn which values will optimally replace missing values by training neural nets and building models to generate the best replacement values. You can then use a Filler node to replace blanks with this value. Note that at least one model is required for each field whose values will be replaced and values should be replaced only from models with sufficient accuracy. This option is time consuming, but if the replacement values for each field are good, it will improve the overall modeling.

**CLEM Functions for Missing Values**

There are several CLEM functions used to handle missing values. The following functions are often used in Select and Filler nodes to discard or fill missing values:

- @BLANK(FIELD)
- @NULL(FIELD)
- undef

The @ functions can be used in conjunction with the @FIELD function to identify the presence of blank or null values in one or more fields. The fields can simply be flagged when blank or null values are present, or they can be filled with replacement values or used in a variety of other operations.
You can use the `undef` function to fill fields with the system-missing value, displayed as `$null$`. For example, to replace any numeric value, you could use a conditional statement, such as:

```plaintext
if not(Age > 17) or not(Age < 66) then undef else Age endif
```

This replaces anything that is not in the range with a system-missing value, displayed as `$null$`. By using the `not()` function, you can catch all other numeric values, including any negatives. For more information, see “Functions Handling Blanks and Null Values” in Appendix A on page 695.

**Note on Discarding Records**

When using a Select node to discard records, note that Clementine syntax uses three-valued logic and automatically includes null values in select statements. To exclude null values (system-missing) in a select CLEM expression, you must
explicitly specify this using and not in the expression. For example, to select and include all records where the type of prescription drug is Drug C, you would use the following select statement:

\[ \text{Drug} = 'drugC' \text{ and not}(@\text{NULL}(\text{Drug})) \]

Earlier versions of Clementine excluded null values in such situations.
Graph Nodes

Chapter 10

Graph Nodes Overview

Several phases of the data mining process use graphs and charts to explore data brought into Clementine. For example, you can connect a Plot or Distribution node to a data source to gain insight into data types and distributions. You can then perform record and field manipulations to prepare the data for downstream modeling operations. Another common use of graphs is to check the distribution and relationships between newly derived fields.

Figure 10-1
Graphs palette

The Graphs palette contains the following nodes:

- Plot
- Multiplot
- Distribution
- Histogram
- Collection
- Web
- Evaluation
Once you have selected fields to examine and specified other options, you can execute the graph node either from within the dialog box or as part of the stream. In the generated graph window, you have the option to generate various types of Derive nodes (Set and Flag) and Select nodes based on a selection or region of data. This feature is extremely powerful because it enables you to visually subset your data (for example, to exclude outliers).

**Overlay Graphs**

A wide variety of overlays are available for graphs in Clementine. Each of these can be used to explore data from different perspectives. Available overlays and applicable graphs are:

- **Color**—plot, histogram, collection
- **Panel**—plot, multiplot, histogram, collection
- **Size**—plot
- **Shape**—plot
- **Transparency**—plot
- **Animation**—multiplot, histogram, collection

*Figure 10-2*

*Graph with size overlay*
Figure 10-3
Graph with panel overlay

Figure 10-4
Graph with color overlay
3-D Graphs

Plots and collection graphs in Clementine have the ability to display information on a third axis. This provides you with additional flexibility when visualizing your data in order to select subsets or derive new fields for modeling.
There are two ways of creating 3-D graphs in Clementine: plotting information on a third axis (true 3-D graphs) and displaying graphs with 3-D effects. Both methods are available for plots and collections.

**To plot information on a third axis:**

- In the graph node dialog box, click the Plot tab.
- Click the 3-D button to enable options for the \( z \) axis.
- Use the Field Chooser button to select a field for the \( z \) axis. In some cases, only symbolic fields are allowed here. The Field Chooser will display the appropriate fields.

**To add 3-D effects to a graph:**

- Once you have created a graph, click the Graph tab in the output window.
- Click the 3-D button to switch the view to a three-dimensional graph.
Animation

Plots, multiplots, and histograms can be “animated” in Clementine. An animation graph works like a movie clip—click the play button to flip through charts for all categories. An animation variable with many categories works especially well, since the animation “flips through” all of the graphs for you. Keeping the number of distinct categories reasonable (such as 15) will ensure normal performance of the software.

Figure 10-7
Animated plot using a variable with three categories

Once you have generated an animated chart, you can use the animation tools in a number of ways:

- Pause the animation at any point.
- Use the slider to view the animation at the desired point (category).

Building Graphs

Once added to a stream, each graph node can be double-clicked to open a tabbed dialog box for specifying options. Most graphs contain a number of unique options presented on one or more tabs. There are also several tab options common to all graphs. The following topics contain more information about these common options.

Setting Output Options for Graphs

For all graph types, you can specify the following options for the filename and display of generated graphs. Note: For distributions, the file types are different and reflect the distribution's similarity to tables. For more information, see “Output Options for the Distribution Node” on page 261.
Output to screen. Select to generate and display the graph in a Clementine window.

Output to file. Select to save the generated graph as a file of the type specified in the File type drop-down list.

File type. Available file types are:
- Bitmap (.bmp)
- JPEG (.jpg)
- PNG (.png)
- HTML document (.html)

Note: The above file types are not available for distributions. For more information, see “Output Options for the Distribution Node” on page 261.

Filename. Specify a filename used for the generated graph. Use the ellipsis button (...) to specify a file and location.

Setting Appearance Options for Graphs

For all graphs except distributions, you can specify appearance options either before graph creation or while exploring the already generated graph.
Setting appearance options for graphs

![Setting appearance options for graphs](image)

**Setting for.** Use the drop-down list to select either Title, Caption, or Labels. (Options specified in this group of controls apply to the item you select here.)

**Font.** Use the drop-down list to specify font type and size. You can also choose to make the font bold or italic using the corresponding buttons.

**Text.** Enter the text used for either a title or caption (specified above using the Setting for drop-down list).

**X label.** Available for Labels only, you can select Custom to specify a custom label for the x axis.

**Y label.** Available for Labels only, you can select Custom to specify a custom label for the y axis.

**Z label.** Available for Labels only in 3-D graphs, you can select Custom to specify a custom label for the z axis.

**Graph background.** Select a color from the drop-down list to use for the graph background. You can specify additional colors by scrolling to the end of the list and selecting Colors.
**Page background.** Select a color from the drop-down list to use for the background of the entire graph window (as opposed to the plot or graph area). You can specify additional colors by scrolling to the end of the list and selecting Colors.

**Display gridline.** Selected by default, this option displays a gridline behind the plot or graph that enables you to more easily determine region and band cutoff points. Gridlines are always displayed in white unless the graph background is white; in this case, they are displayed in gray.

**Symbol Size.** Enter a point size used for display symbols or use the arrows to adjust the default size.

Color settings used for points and bars are specified in the User Options dialog box.

To access this dialog box, from the Clementine window menus, choose:

Tools
  User Options...
Then click the Display tab.

*Note:* Colors used for points, lines, and bars must be specified before graph creation in order for changes to take effect.

**Using Graphs**

Once you have created graphs, there are several ways to customize and manipulate them. You can explore graphs in any of the following ways:

- Use the mouse to select an area of a graph for further operations.
- Use the options available from the menu bar. Different graphs may have different types of menus and options available.
- Right-click on a selected area to bring up a context menu of available options for that area.

**Figure 10-10**
*Evaluation chart with context-menu options for a defined region*
Using these methods, you can perform the following operations, depending on the type of graph created:

- Highlight data regions on plot graphs using the mouse to specify a rectangular area.
- Highlight data bands on histograms and collection graphs by clicking in the graph area.
- Identify and label subsets of your data.
- Generate manipulation nodes based on selected areas of the graph.

Figure 10-11
Exploring a plot using a variety of methods
Chapter 10

General Graph Window Options

Each graph has a number of options, such as exporting, printing, adding to projects, and publishing to the Web. Some of these options are available from the File menu and others from the toolbar.

Figure 10-12
File menu and toolbar for graph windows

File Menu Options

From the file menu of a graph window, you can perform the following operations:

- Save the graph as a file. Using the Save dialog box, you can also add the file to the currently open project.
- Close the graph window/file.
- Close the graph and delete it from the Output tab.
- Print the graph and set up printing options, including headers and footers.
Export the graph in a number of formats—graphic formats and table or data formats, where applicable.

Publish the graph as an image file to the repository used by SPSS Web Deployment Framework. This makes the graph available to all applications using the framework.

Export the graph as HTML to the desired location.

**Toolbar Options**

Using the toolbar buttons in the graph window, you can perform the following operations:

- Copy the graph to the clipboard for pasting into another application.
- Add the graph file to the current project. *Note:* You will be prompted to save the graph if it is unsaved.
- Print the graph to the default printer without opening any printing dialog boxes.

The remainder of this chapter focuses on the specific options for creating graphs and using them in their output windows.

**Plot Node**

Plot nodes show the relationship between numeric fields. You can create a plot using points (also known as a scatterplot) or you can use lines. You can create three types of line plots by specifying an X Mode in the dialog box.

**X Mode = Sort**

Setting X Mode to Sort causes data to be sorted by values for the field plotted on the x axis. This produces a single line running from left to right on the graph. Using a set variable as an overlay produces multiple lines of different hues running from left to right on the graph.
**X Mode = Overlay**

Setting X Mode to Overlay creates multiple line plots on the same graph. Data are not sorted for an overlay plot; as long as the values on the $x$ axis increase, data will be plotted on a single line. If the values decrease, a new line begins. For example, as $x$ moves from 0 to 100, the $y$ values will be plotted on a single line. When $x$ falls below 100, a new line will be plotted in addition to the first one. The finished plot might have numerous plots useful for comparing several series of $y$ values. This type of plot is useful for data with a periodic time component, such as electricity demand over successive 24-hour periods.
**Figure 10-14**
*Line plot with X Mode set to Overlay*

![Line plot with X Mode set to Overlay](image)

**X Mode = As Read**

Setting X Mode to As Read plots x and y values as they are read from the data source. This option is useful for data with a time series component where you are interested in trends or patterns that depend on the order of the data. You may need to sort the data before creating this type of plot. It may also be useful to compare two similar plots with X Mode set to Sort and As Read in order to determine how much of a pattern depends on the sorting.
Figure 10-15
Line plot shown earlier as Sort, executed again with X Mode set to As Read

Setting Options for the Plot Node

Plots show values of a Y field against values of an X field. Often, these fields correspond to a dependent variable and an independent variable, respectively.
Figure 10-16
Setting options for a Plot node

X field. Select a field from the list to display on the x axis, also known as the horizontal axis or abscissa.

Y field. Select a field from the list to display on the y axis, also known as the vertical axis or ordinate.

Z field. When you click the 3-D chart button, a third field becomes available for you to select a field from the list to display on the z axis.

Overlay. There are several ways to illustrate categories for data values. For example, you can use maincrop as a color overlay to indicate the estincome and claimvalue values for the main crop grown by claim applicants.

- Color. Select a field to illustrate categories for data values by using a different color for each value.
- Panel. Select a set or flag field to use in making a separate chart for each category. Charts will be “paneled,” or displayed together in one output window.
- Size. Select a field to illustrate categories for data values by using a gradient of sizes. This overlay is not available for line plots.
Chapter 10

- **Animation.** Select a set or flag field to illustrate categories for data values by creating a series of charts displayed in sequence using animation.

- **Shape.** Select a set or flag field to illustrate categories for data values by using a different point shape for each category. This overlay is not available for line plots.

- **Transparency.** Select a field to illustrate categories for data values by using a different level of transparency for each category. This overlay is not available for line plots.

When using a range field as an overlay for color, size, and transparency, the legend uses a continuous scale rather than discrete categories.

**Overlay function.** Select to specify a known function to compare to actual values. For example, to compare actual versus predicted values, you can plot the function \( y = x \) as an overlay. Specify a function for \( y = x \) in the text box. The default function is \( y = x \), but you can specify any sort of function, such as a quadratic function or an arbitrary expression, in terms of \( x \). If you have specified a 3-D graph, you can also specify an overlay function for \( z \). *Note:* Overlay functions are not available for a panel or animation graph.

Once you have set options for a plot, you can execute the plot directly from the dialog box by clicking **Execute**. You may, however, want to use the **Options** tab for additional specifications, such as binning, X Mode, and style.
**Additional Plot Options**

**Figure 10-17**
*Options tab settings for a Plot node*

![Plot node options settings](image)

**Style.** Select either Point or Line for the plot style. Selecting Point activates the Point Type control described below. Selecting Line activates the X Mode control described below.

**Point Type.** Controls the shape of points in a scatterplot. By default, the point shape is a plus symbol (+). Use the drop-down list to select a different shape, such as a hexagon, triangle, vertical dash, or dot. Once the graph is created, you cannot change the point shape but you can alter its size.

**X Mode.** For line plots, you must choose an X Mode to define the style of the line plot. Select Sort, Overlay, or As Read. For Overlay or As Read, you should specify a maximum data set size used to sample the first $n$ records. Otherwise, the default 2000 records will be used.

**Automatic X range.** Select to use the entire range of values in the data along the $x$ axis. Deselect to use an explicit subset of values based on your specified Min and Max values. Either enter values or use the arrows. Automatic ranges are selected by default to enable rapid graph building.
**Automatic Y range.** Select to use the entire range of values in the data along the y axis. Deselect to use an explicit subset of values based on your specified Min and Max values. Either enter values or use the arrows. Automatic ranges are selected by default to enable rapid graph building.

**Automatic Z range.** When a 3-D graph is specified on the Plot tab, you can select this option to use the entire range of values in the data along the z axis. Deselect to use an explicit subset of values based on your specified Min and Max values. Either enter values or use the arrows. Automatic ranges are selected by default to enable rapid graph building.

**Agitation (+/–).** Also known as jittering, agitation is useful for point plots of a data set in which many values are repeated. In order to see a clearer distribution of values, you can distribute the points randomly around the actual value. Set the agitation value to add random error that will jitter the points in axis coordinates. Each point will be jittered by jitterFactor x (randomVal −0.5), where randomVal >= 0 and <= 1. A value of 0.2 (corresponding to a maximum of 10% of the frame real estate) works well here.

*Note to users of earlier versions of Clementine:* The agitation value specified in a plot uses a different metric in this release of Clementine. In earlier versions, the value was an actual number, but it is now a proportion of the frame size. This means that agitation values in old streams are likely to be too large. For this release, any nonzero agitation values will be converted to the value 0.2.

**When number of records greater than.** Specify a method for plotting large data sets. You can specify a maximum data set size or use the default 2000 records. Performance is enhanced for large data sets when you select the Bin or Sample options. Alternatively, you can choose to plot all data points by selecting Use all data, but you should note that this may dramatically decrease the performance of the software. *Note:* When X Mode is set to Overlay or As Read, these options are disabled and only the first n records are used.

- **Bin.** Select to enable binning when the data set contains more than the specified number of records. Binning divides the graph into fine grids before actually plotting and counts the number of points that would appear in each of the grid cells. In the final graph, one point is plotted per cell at the bin centroid (average of all point locations in the bin). The size of the plotted symbols indicates the number of points in that region (unless you have used size as an overlay). Using the centroid and size to represent the number of points makes the binned plot a superior way to represent large data sets because it prevents overplotting in
dense regions (undifferentiated masses of color) and reduces symbol artifacts (artificial patterns of density). Symbol artifacts occur when certain symbols (particularly the plus symbol [+]) collide in a way that produces dense areas not present in the raw data.

- **Sample.** Select to randomly sample the data to the number of records entered in the text field. 2000 records is the default.

**Using a Plot Graph**

Plots, multiplots, and evaluation charts are essentially plots of \( X \) against \( Y \). For example, if you are exploring potential fraud in agricultural grant applications (as illustrated in `fraud.str` in the `demos` folder of your Clementine installation), you might want to plot the income claimed on the application versus the income estimated by a neural net. Using an overlay, such as crop type, will illustrate whether there is a relationship between claims (value or number) and type of crop.

**Figure 10-18**
Plot of the relationship between estimated income and claim value with main crop type as an overlay
Since plots, multiplots, and evaluation charts are two-dimensional displays of $Y$ against $X$, it is easy to interact with them by selecting regions with the mouse. A region is an area of the graph described by its minimum and maximum $X$ and $Y$ values. *Note:* Regions cannot be defined in 3-D or animated plots.

**To define a region:**

You can either use the mouse to interact with the graph, or you can use the Edit Graph Regions dialog box to specify region boundaries and related options. For more information, see “Editing Graph Regions” on page 256. To use the mouse for defining a region:

- Click the left mouse button somewhere in the plot to define a corner of the region.
- Drag the mouse to the position desired for the opposite corner of the region. The resulting rectangle cannot exceed the boundaries of the axes.
- Release the mouse button to create a permanent rectangle for the region. By default, the new region is called $Region<N>$, where $N$ corresponds to the number of regions already created in the Clementine session.
Once you have defined a region, there are numerous ways to delve deeper into the selected area of the graph. Use the mouse in the following ways to produce feedback in the graph window:

- Hover over data points to provide point-specific information.
- Right-click and hold the mouse button in a region to provide information about boundaries of that region.
- Simply right-click in a region to bring up a context menu with additional options, such as generating process nodes.
Chapter 10

Figure 10-20
Exploring the region of high claim values

To rename a region:

► Right-click anywhere in the defined region.

► From the context menu, choose Rename Region.

► Enter a new name and click OK.

*Note:* You can also rename the default region by right-clicking anywhere outside the region and choosing Rename Default Region.

To delete a region:

► Right-click anywhere in the defined region.
From the context menu, choose Delete Region.

Once you have defined regions, you can select subsets of records on the basis of their inclusion in a particular region or in one of several regions. You can also incorporate region information for a record by producing a Derive node to flag records based on their inclusion in a region.

**To select or flag records in a single region:**

- Right-click in the region. Note that when you hold the right mouse button, the details for the region are displayed in the feedback panel below the plot.

- From the context menu, choose Generate Select Node for Region or Generate Derive Node for Region.

A Select node or Derive node is automatically added to the stream canvas with the appropriate options and conditions specified. The Select node selects all records in the region. The Derive node generates a flag for records whose values fall within the region. The flag field name corresponds to the region name, with the flags set to $T$ for records inside the region and $F$ for records outside.

**To select, flag, or derive a set for records in all regions:**

- From the Generate menu in the graph window, choose Derive Node (Set), Derive Node (Flag), or Select Node.

- For all selections, a new node appears on the stream canvas with the following characteristics, depending on your selection:
  - **Derive Set.** Produces a new field called region for each record. The value of that field is the name of the region into which the records fall. Records falling outside all regions receive the name of the default region. (Right-click outside all regions and choose Rename Default Region to change the name of the default region.)
  - **Derive Flag.** Creates a flag field called in_region with the flags set to $T$ for records inside any region and $F$ for records outside all regions.
  - **Select Node.** Generates a new node that tests for inclusion in any region. This node selects records in any region for downstream processing.
Chapter 10

**Editing Graph Regions**

For plots, multiplots, and evaluation charts, you can edit the properties of regions defined on the graph. To open this dialog box, from graph window menus, choose:

Edit

Graph Regions...

*Figure 10-21*

*Specifying properties for the defined regions*

<table>
<thead>
<tr>
<th>Region Name</th>
<th>Min X</th>
<th>Max X</th>
<th>Min Y</th>
<th>Max Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>region5</td>
<td>92.2007107...</td>
<td>177.0363.80...</td>
<td>332563.929...</td>
<td>6077670.451...</td>
</tr>
<tr>
<td>region6</td>
<td>270336.716...</td>
<td>753287.855...</td>
<td>58336.3753...</td>
<td>177740.305...</td>
</tr>
</tbody>
</table>

**Region Name.** Enter adjustments to the defined region names.

- You can manually specify the boundaries of the region by adjusting the Min and Max values for X and Y.
- Add new regions by specifying the name and boundaries. Then press the Enter key to begin a new row.
- Delete regions by selecting one in the table and clicking the delete button.

**Multiplot Node**

A multiplot is a special type of plot that displays multiple Y fields over a single X field. The Y fields are plotted as colored lines and each is equivalent to a Plot node with Style set to Line and X Mode set to Sort. Multiplots are useful when you have time sequence data and want to explore the fluctuation of several variables over time.
Setting Options for the Multiplot Node

**X field.** Select a field to display along the x axis.

**Y fields.** Select one or more fields from the list to display over the range of X field values. Use the Field Chooser button to select multiple fields. Click the delete button to remove fields from the list.

**Overlay.** There are several ways to illustrate categories for data values. For example, you might use an animation overlay to display multiple plots for each value in the data. This is useful for sets with many categories, such as 10. When used for sets with more than 15 categories, you may notice a decrease in performance.

- **Panel.** Select a set or flag field to use in making a separate chart for each category. Charts will be “paneled,” or displayed together in one output window.
- **Animation.** Select a set or flag field to illustrate categories for data values by creating a series of charts displayed in sequence using animation.
**Normalize.** Select to scale all $Y$ values to the range 0–1 for display on the graph. Normalizing helps you explore the relationship between lines that might otherwise be obscured in the graph.

**Figure 10-23**
*Standard multiplot showing power-plant fluctuation over time (note that without normalizing, the plot for Pressure is impossible to see)*

**Figure 10-24**
*Normalized multiplot showing a plot for Pressure*
**Overlay function.** Select to specify a known function to compare to actual values. For example, to compare actual versus predicted values, you can plot the function $y = x$ as an overlay. Specify a function in the $y = \text{text box}$. The default function is $y = x$, but you can specify any sort of function, such as a quadratic function or an arbitrary expression, in terms of $x$.

**When number of records greater than.** Specify a method for plotting large data sets. You can specify a maximum data set size or use the default 2000 points. Performance is enhanced for large data sets when you select the Bin or Sample options. Alternatively, you can choose to plot all data points by selecting Use all data, but you should note that this may dramatically decrease the performance of the software. *Note:* When X Mode is set to Overlay or As Read, these options are disabled and only the first $n$ records are used.

- **Bin.** Select to enable binning when the data set contains more than the specified number of records. Binning divides the graph into fine grids before actually plotting and counts the number of connections that would appear in each of the grid cells. In the final graph, one connection is used per cell at the bin centroid (average of all connection points in the bin).

- **Sample.** Select to randomly sample the data to the specified number of records.

**Using a Multiplot Graph**

Plots and multiplots are two-dimensional displays of $Y$ against $X$, making it easy to interact with them by selecting regions with the mouse. A region is an area of the graph described by its minimum and maximum $X$ and $Y$ values.

Since multiplots are essentially a type of plot, the graph window displays the same options as those for the Plot node. For more information, see “Using a Plot Graph” on page 251.

**Distribution Node**

A distribution graph shows the occurrence of symbolic (non-numeric) values, such as mortgage type or gender, in a data set. A typical use of the Distribution node is to show imbalances in the data that can be rectified by using a Balance node before creating a model. You can automatically generate a Balance node using the Generate menu in a distribution graph window.
Note: To show the occurrence of numeric values, you should use a Histogram node.

Figure 10-25
Setting options for a Distribution node

Setting Options for the Distribution Node

Plot. Select the type of distribution. Select Selected fields to show the distribution of the selected field. Select All flags (true values) to show the distribution of true values for flag fields in the data set.

Field. Select a set or flag field for which to show the distribution of values. Only fields that have not been explicitly set as numeric appear on the list.

Overlay. Select a set or flag field to use as a color overlay, illustrating the distribution of its values within each value of the field selected above. For example, you can use marketing campaign response (pep) as an overlay for number of children (children) to illustrate responsiveness by family size.

Normalize by color. Select to scale bars such that all bars take up the full width of the graph. The overlay values equal a proportion of each bar, making comparisons across categories easier.
Sort. Select the method used to display values in the distribution graph. Select Alphabetic to use alphabetical order or By count to list values in decreasing order of occurrence.

Proportional scale. Select to scale the distribution of values such that the value with the largest count fills the full width of the plot. All other bars are scaled against this value. Deselecting this option scales bars according to the total counts of each value.

Output Options for the Distribution Node

Options displayed on the Output tab for distributions are slightly different than for other graphs.

Output to screen. Select to generate and display the graph in a Clementine window.

Output to file. Select to save the generated graph as a file of the type specified in the File type drop-down list.

Filename. Specify a filename used for the generated graph. Use the ellipses button (…) to specify a specific file and location.

File type. Available file types are:
- Formatted (.tab)
- Data (comma delimited) (.dat)
- HTML document (.html)

Lines per page. When saving output as HTML, this option is enabled to allow you to determine the length of each HTML page. The default setting is 400.

Using a Distribution Graph

Distribution nodes are used to show the distribution of symbolic values in a data set. They are frequently used before manipulation nodes to explore the data and correct any imbalances. For example, if instances of respondents without children occur much more frequently than other types of respondents, you might want to reduce these instances so that a more useful rule can be generated in later data mining operations. A Distribution node will help you to examine and make decisions about such imbalances.
Once you have created a distribution graph and examined the results, you can use options from the menus to group values, copy values, and generate a number of nodes for data preparation.

**Edit Menu Options**

You can use options on the Edit menu to group, select, and copy values in the distribution table.

**To select and copy values from a distribution:**

- Click and hold the mouse button while dragging it to select a set of values. You can use the Edit menu to Select All values.
- From the Edit menu, choose Copy or Copy (Inc. field names).
- Paste to the clipboard or into the desired application.

*Note:* The bars do not get copied directly. Instead, the table values are copied. This means that overlaid values will not be displayed in the copied table.

**To group values from a distribution:**

- Select values for grouping using the Ctrl-click method.
- From the Edit menu, choose Group.
You can also:

- Ungroup values by selecting the group name in the distribution list and choosing Ungroup from the Edit menu.
- Edit groups by selecting the group name in the distribution list and choosing Edit group from the Edit menu. This opens a dialog box where values can be shifted to and from the group.

Figure 10-27
Edit group dialog box

*Generate Menu Options*

You can use options on the Generate menu to select a subset of data, derive a flag field, regroup values, or balance the data. These operations generate a data preparation node and place it on the stream canvas. To use the generated node, connect it to an existing stream.

- **Select Node.** Select any cell from the graph to generate a Select node for that category. You can select multiple categories using Ctrl-click in the distribution table.
- **Derive Node.** Select any cell from the graph to generate a Derive flag node for that category. You can select multiple categories using Ctrl-click in the distribution table.
- **Balance Node (boost).** Use this option to generate a Balance node that boosts the size of smaller subsets.
- **Balance Node (reduce).** Use this option to generate a Balance node that reduces the size of larger subsets.
Reclassify Node (groups). Use this option to generate a Reclassify node that recodes specific values of the displayed field depending upon their inclusion in a group. Groups can be selected using the Ctrl-click method. You can group values by selecting them and using the Edit menu options.

Reclassify Node (values). Use this option to generate a blank Reclassify node. Values can be selected for recoding from the Reclassify node dialog box.

Histogram Node

Histogram nodes show the occurrence of values for numeric fields. They are often used to explore the data before manipulations and model building. Similar to the Distribution node, histograms are frequently used to reveal imbalances in the data.

*Note:* To show the occurrence of values for symbolic fields, you should use a Distribution node.

Figure 10-28
Setting options for a Histogram node

Field. Select a numeric field for which to show the distribution of values. Only fields that have not been explicitly defined as symbolic (categorical) will be listed.
**Overlay.** Select a symbolic field to show categories of values for the field selected above. Selecting an overlay field converts the histogram to a stacked chart with colors used to represent different categories of the overlay field. Three types of overlays are available for histograms:

- **Color.** Select a field to illustrate categories for data values by using a different color for each value.
- **Panel.** Select a set or flag field to use in making a separate graph for each category. Graphs will be “paneled,” or displayed together in one output window.
- **Animation.** Select a set or flag field to illustrate categories for data values by creating a series of graphs displayed in sequence using animation.

**Setting Additional Options for the Histogram Node**

Figure 10-29
*Options tab settings for a Histogram node*

**Automatic X range.** Select to use the entire range of values in the data along the $x$ axis. Deselect to use an explicit subset of values based on your specified Min and Max values. Either enter values or use the arrows. Automatic ranges are selected by default to enable rapid graph building.
Bins. Select By number to display a fixed number of histogram bars whose width depends on the range specified above and the number of buckets specified below. Select By width to create a histogram with bars of a fixed width (specified below). The number of bars depends on the specified width and the range of values.

No. of bins. Specify the number of buckets (bars) to be used in the histogram. Use the arrows to adjust the number.

Bin width. Specify the width of histogram bars.

Normalize by color. Select to adjust all bars to the same height, displaying overlaid values as a percentage of the total cases in each bar.

Separate bands for each color. Select to display each overlaid value as a separate band on the graph.

Using Histograms and Collections

Histograms and collections offer a similar window into your data before modeling.

- Histograms show the distribution of values in a numeric field whose values range along the x axis.

- Collections show the distribution of values for one numeric field relative to the values of another, rather than the occurrence of values for a single field.

Both types of charts are frequently used before manipulation nodes to explore the data and correct any imbalances by generating a Balance node from the output window. You can also generate a Derive Flag node to add a field showing which band each record falls into or a Select node to select all records within a particular set or range of values. Such operations help you to focus on a particular subset of data for further exploration.
Several options are available in the histogram window. These options apply for both histograms and collections. For example, you can:

- Split the range of values on the $x$ axis into **bands**.
- Generate a Select or Derive Flag node based on inclusion in a particular band's range of values.
- Generate a Derive Set node to indicate the band into which a record's values fall.
- Generate a Balance node to correct imbalances in the data.
- View the graph in 3-D (available for collections only).
To define a band:

You can either use the mouse to interact with the graph, or you can use the Edit Graph Bands dialog box to specify the boundaries of bands and other related options. For more information, see “Editing Graph Bands” on page 271. To use the mouse for defining a band:

- Click anywhere in the histogram to set a line defining a band of values.
- Or, click the Bands button on the toolbar to split the graph into equal bands. This method adds additional options to the toolbar, which you can use to specify a number of equal bands.

Figure 10-31
Creating equal bands
Once you have a defined a band, there are numerous ways to delve deeper into the selected area of the graph. Use the mouse in the following ways to produce feedback in the graph window:

- Hover over bars to provide bar-specific information.
- Check the range of values for a band by right-clicking inside a band and reading the feedback panel at the bottom of the window.
- Simply right-click in a band to bring up a context menu with additional options, such as generating process nodes.
- Rename bands by right-clicking in a band and selecting Rename Band. By default, bands are named \textit{bandN}, where \textit{N} equals the number of bands from left to right on the \textit{x} axis.
- Move the boundaries of a band by selecting a band line with your mouse and moving it to the desired location on the \textit{x} axis.
- Delete bands by right-clicking on a line and selecting Delete Band.

Once you have created a histogram, defined bands, and examined the results, you can use options on the Generate menu and the context menu to create Balance, Select, or Derive nodes.
To select or flag records in a particular band:

- Right-click in the band. Notice that the details for the band are displayed in the feedback panel below the plot.

- From the context menu, choose Generate Select Node for Band or Generate Derive Node for Band.

A Select node or Derive node is automatically added to the stream canvas with the appropriate options and conditions specified. The Select node selects all records in the band. The Derive node generates a flag for records whose values fall within the
band. The flag field name corresponds to the band name, with flags set to T for records inside the band and F for records outside.

**To derive a set for records in all regions:**

- From the Generate menu in the graph window, choose Derive Node.
- A new Derive Set node appears on the stream canvas with options set to create a new field called `band` for each record. The value of that field equals the name of the band that each record falls into.

**To create a Balance node for imbalanced data:**

- From the Generate menu in the graph window, choose one of the two Balance node types:
  - **Balance Node (boost).** Generates a Balance node to boost the occurrence of infrequent values.
  - **Balance Node (reduce).** Generates a Balance node to reduce the frequency of common values.

The generated node will be placed on the stream canvas. To use the node, connect it to an existing stream.

**Editing Graph Bands**

For histograms, collections, and evaluation charts, you can edit the properties of bands defined on the graph. To open this dialog box, from graph window menus, choose:

Edit
Graph Bands...
**Band Name.** Enter adjustments to the defined band names.

- You can manually specify the boundaries of the band by adjusting the Min and Max values for X and Y.
- Add new bands by specifying the name and boundaries. Then press the Enter key to begin a new row.
- Delete bands by selecting one in the table and clicking the delete button.

**Collection Node**

Collections are similar to histograms except that collections show the distribution of values for one numeric field relative to the values of another, rather than the occurrence of values for a single field. A collection is useful for illustrating a variable or field whose values change over time. Using 3-D graphing, you can also include a symbolic axis displaying distributions by category.
**Collect.** Select a field whose values will be collected and displayed over the range of values for the field specified below in **Over.** Only fields that have not been defined as symbolic are listed.

**Over.** Select a field whose values will be used to display the collection field specified above.

**By.** Enabled when creating a 3-D graph, this option allows you to select a set or flag field used to display the collection field by categories.

**Operation.** Select what each bar or bucket in the collection graph represents. Options include **Sum, Mean, Max, Min,** and **Standard Deviation.**

**Overlay.** Select a symbolic field to show categories of values for the field selected above. Selecting an overlay field converts the collection and creates multiple bars of varying colors for each category. Three types of overlays are available for collections:

- **Color.** Select a field to illustrate categories for data values by using a different color for each value.
- **Panel.** Select a set or flag field to use in making a separate graph for each category. Graphs will be “paneled,” or displayed together in one output window.
- **Animation.** Select a set or flag field to illustrate categories for data values by creating a series of graphs displayed in sequence using animation.
Setting Additional Options for the Collection Node

**Automatic X range.** Select to use the entire range of values in the data along the $x$ axis. Deselect to use an explicit subset of values based on your specified Min and Max values. Either enter values or use the arrows. Automatic ranges are selected by default to enable rapid graph building.

**Bins.** Select By number to display a fixed number of collection bars whose width depends on the range specified above and the number of buckets specified below. Select By width to create a collection with bars of a fixed width (specified below). The number of bars depends on the specified width and the range of values.

**No. of bins.** Specify the number of buckets (bars) to be used in the collection. Use the arrows to adjust the number.

**Bin width.** Specify the width of collection bars.

Using a Collection Graph

Collection nodes show the distribution of values in a numeric field whose values range along the $x$ axis. They are frequently used before manipulation nodes to explore the data and correct any imbalances by generating a Balance node from the graph window. You can also generate a Derive Flag node to add a field showing which range (band) each record falls into or a Select node to select all records within a particular range of values. Such operations help you to focus on a particular subset of data for further exploration.
Once you have created a collection graph, several options are available in the graph window. For example, you can:

- Split the range of values on the \( x \) axis into bands.
Generate a Select or Derive Flag node based on inclusion in a particular band's range of values.

Generate a Derive Set node to indicate which band contains a record's value.

Generate a Balance node to correct imbalances in the data.

Figure 10-37
Options for generating Select and Derive nodes to examine a band of interest

Since collections are very similar to histograms, the graph window displays the same options. For more information, see “Using Histograms and Collections” on page 266.

**Web Node**

Web nodes show the strength of relationships between values of two or more symbolic fields. The graph displays connections using varying types of lines to indicate connection strength. You can use a Web node, for example, to explore the relationship between the purchase of various items at an e-commerce site or a traditional retail outlet.
Directed Webs

Directed Web nodes are similar to Web nodes in that they show the strength of relationships between symbolic fields. However, directed web graphs show connections only from one or more From fields to a single To field. The connections are unidirectional in the sense that they are one-way connections.

Figure 10-39
Directed web showing the relationship between the purchase of grocery items and gender
Like Web nodes, the graph displays connections using varying types of lines to indicate connection strength. You can use a Directed Web node, for example, to explore the relationship between gender and a proclivity for certain purchase items.

**Setting Options for the Web Node**

**Figure 10-40**  
*Setting options for a Web node*

**Web.** Select to create a web graph illustrating the strength of relationships between all specified fields.

**Directed web.** Select to create a directional web graph illustrating the strength of relationships between multiple fields and the values of one field, such as gender or religion. When this option is selected, a **To Field** is activated and the **Fields control Field** below is renamed **From Fields** for additional clarity.
**To Field (directed webs only)**. Select a flag or set field used for a directed web. Only fields that have not been explicitly set as numeric are listed.

**Fields/From Fields**. Select fields to create a web graph. Only fields that have not been explicitly set as numeric are listed. Use the Field Chooser button to select multiple fields or select fields by type. *Note*: For a directed web, this control is used to select From fields.

**Show true flags only**. Select to display only true flags for a flag field. This option simplifies the web display and is often used for data where the occurrence of positive values is of special importance.

**Line values are**. Select a threshold type from the drop-down list.

- Absolute sets thresholds based on the number of records having each pair of values.
- Overall percentages shows the absolute number of cases represented by the link as a proportion of all of the occurrences of each pair of values represented in the web plot.
- Percentages of smaller field/value and Percentages of larger field/values indicate which field/value to use for evaluating percentages. For example, suppose 100 records have the value *drugY* for the field *Drug* and only 10 have the value *LOW* for the field *BP*. If seven records have both values *drugY* and *LOW*, this percentage is either 70% or 7%, depending on which field you are referencing, smaller (*BP*) or larger (*Drug*).

*Note*: For directed web graphs, the third and fourth options above are not available. Instead, you can select Percentage of “To” field/value and Percentage of “From” field/value.

**Strong links are heavier**. Selected by default, this is the standard way of viewing links between fields.
**Weak links are heavier.** Select to reverse the meaning of links displayed in bold lines. This option is frequently used for fraud detection or examination of outliers.

**Setting Additional Options for the Web Node**

The Options tab for Web nodes contains a number of additional options to customize the output graph.

**Figure 10-42**
*Options tab settings for a Web node*

**Number of Links.** The following controls are used to control the number of links displayed in the output graph. Some of these options, such as Weak links above and Strong links above, are also available in the output graph window. You can also use a slider control in the final graph to adjust the number of links displayed.

- **Maximum number of links to display.** Specify a number indicating the maximum number of links to show on the output graph. Use the arrows to adjust the value.
- **Show only links above.** Specify a number indicating the minimum value for which to show a connection in the web. Use the arrows to adjust the value.

- **Show all links.** Specify to display all links regardless of minimum or maximum values. Selecting this option may increase processing time if there are a large number of fields.

**Discard if very few records.** Select to ignore connections that are supported by too few records. Set the threshold for this option by entering a number in Min. records/line.

**Discard if very many records.** Select to ignore strongly supported connections. Enter a number in Max. records/line.

- **Strong links above.** Specify a threshold for strong connections (heavy lines) and regular connections (normal lines). All connections above this value are considered strong.

- **Weak links below.** Specify a number indicating the threshold for weak connections (dotted lines) and regular connections (normal lines). All connections below this value are considered weak.

- **Link Size.** Specify options for controlling the size of links:
  - **Link size varies continuously.** Select to display a range of link sizes reflecting the variation in connection strengths based on actual data values.
  - **Link size shows strong/normal/weak categories.** Select to display three strengths of connections—strong, normal, and weak. The cut-off points for these categories can be specified above as well as in the final graph.

- **Web Display.** Select a type of web display:
  - **Circle.** Select to use the standard web display.
  - **Network layout.** Select to use an algorithm to group together the strongest links. This is intended to highlight strong links using spatial differentiation as well as weighted lines.
Figure 10-43
Network display showing strong connections from frozenmeal and cannedveg to other grocery items

**Appearance Options for the Web Plot**

The Appearance tab for web plots contains a subset of options available for other types of graphs.
**Figure 10-44**  
Appearance tab settings for a web plot

![Graph Nodes](image)

**Setting for.** Use the drop-down list to select either Title or Caption. (Options specified in this control apply to the selected item.)

**Font.** Use the drop-down list to specify font type and size. You can also choose to make the font bold or italic using the corresponding buttons.

**Text.** Enter the text used for either a title or caption (specified above using the Setting for drop-down list).

**Graph background.** Select a color from the drop-down list. You can specify additional colors by scrolling to the end of the list and selecting Colors.

**Symbol Size.** Enter a size used for display symbols, or use the arrows to adjust the default size. Increasing this number will result in larger symbols.
Using a Web Graph

Web nodes are used to show the strength of relationships between values of two or more symbolic fields. Connections are displayed in a graph with varying types of lines to indicate connections of increasing strength. You can use a Web node, for example, to explore the relationship between cholesterol levels, blood pressure, and the drug that was effective in treating the patient's illness.

- Strong connections are shown with a heavy line. This indicates that the two values are strongly related and should be further explored.
- Medium connections are shown with a line of normal weight.
- Weak connections are shown with a dotted line.
- If no line is shown between two values, this means either that the two values never occur in the same record or that this combination occurs in a number of records below the threshold specified in the Web node dialog box.

Once you have created a Web node, there are several options for adjusting the graph display and generating nodes for further analysis.
Figure 10-45
Web graph indicating a number of strong relationships, such as normal blood pressure with DrugX and high cholesterol with DrugY

For both Web nodes and Directed Web nodes, you can:

- Change the layout of the web display.
- Hide points to simplify the display.
- Change the thresholds controlling line styles.
- Highlight lines between values to indicate a “selected” relationship.
- Generate a Select node for one or more “selected” records or a Derive Flag node associated with one or more relationships in the web.

To adjust points:

- Move points by clicking the mouse on a point and dragging it to the new location. The web will be redrawn to reflect the new location.
Chapter 10

- **Hide** points by right-clicking on a point in the web and choosing Hide or Hide and Replan from the context menu. Hide simply hides the selected point and any lines associated with it. Hide and Replan redraws the web, adjusting for any changes you have made. Any manual moves are undone.

- **Show** all hidden points by choosing Reveal All or Reveal All and Replan from the Web menu in the graph window. Selecting Reveal All and Replan redraws the web, adjusting to include all previously hidden points and their connections.

**To select, or “highlight,” lines:**

- Left-click to select a line and highlight it in red.

- Continue to select additional lines by repeating this process.

You can deselect lines by choosing Clear Selection from the Web menu in the graph window.

**To view the web using a network layout:**

- From the Web menu, choose Network.

- To return to circle layout, select Circle from the same menu.

**To select or flag records for a single relationship:**

- Right-click on the line representing the relationship of interest.

- From the context menu, choose Generate Select Node For Link or Generate Derive Node For Link.

A Select node or Derive node is automatically added to the stream canvas with the appropriate options and conditions specified:

- The Select node selects all records in the given relationship.

- The Derive node generates a flag indicating whether the selected relationship holds true for records in the entire data set. The flag field is named by joining the two values in the relationship with an underscore, such as $LOW_{drugC}$ or $drugC_{LOW}$. 
To select or flag records for a group of relationships:

- Select the line(s) in the web display representing relationships of interest.
- From the Generate menu in the graph window, choose Select Node (“And”), Select Node (“Or”), Derive Node (“And”), or Derive Node (“Or”).
  - The “Or” nodes give the disjunction of conditions. This means that the node will apply to records for which any of the selected relationships hold.
  - The “And” nodes give the conjunction of conditions. This means that the node will apply only to records for which all selected relationships hold. An error occurs if any of the selected relationships are mutually exclusive.

After you have completed your selection, a Select node or Derive node is automatically added to the stream canvas with the appropriate options and conditions specified.

Adjusting Web Thresholds

After you have created a web graph, you can adjust the thresholds controlling line styles using the toolbar slider to change the minimum visible line. You can also view additional threshold options by clicking the yellow double-arrow button on the toolbar to expand the web graph window. Then click the Controls tab to view additional options.
Threshold values are. Shows the type of threshold selected during creation in the Web node dialog box.

**Strong links are heavier.** Selected by default, this is the standard way of viewing links between fields.

**Weak links are heavier.** Select to reverse the meaning of links displayed in bold lines. This option is frequently used for fraud detection or examination of outliers.

**Web Display.** Specify options for controlling the size of links in the output graph:

- **Size varies continuously.** Select to display a range of link sizes reflecting the variation in connection strengths based on actual data values.

- **Size shows strong/normal/weak categories.** Select to display three strengths of connections—strong, normal, and weak. The cutoff points for these categories can be specified above as well as in the final graph.

**Strong links above.** Specify a threshold for strong connections (heavy lines) and regular connections (normal lines). All connections above this value are considered strong. Use the slider to adjust the value or enter a number in the field.
**Weak links below.** Specify a number indicating the threshold for weak connections (dotted lines) and regular connections (normal lines). All connections below this value are considered weak. Use the slider to adjust the value or enter a number in the field.

After you have adjusted the thresholds for a web, you can replan, or redraw, the web display with the new threshold values by clicking the black replan button on the web graph toolbar. Once you have found settings that reveal the most meaningful patterns, you can update the original settings in the Web node (also called the Parent Web node) by choosing Update Parent Node from the Web menu in the graph window.

**Creating a Web Summary**

You can create a web summary document that lists strong, medium, and weak links by clicking the yellow double-arrow button on the toolbar to expand the web graph window. Then click the Summary tab to view tables for each type of link. Tables can be expanded and collapsed using the toggle buttons for each.
### Evaluation Chart Node

The Evaluation Chart node offers an easy way to evaluate and compare predictive models to choose the best model for your application. Evaluation charts show how models perform in predicting particular outcomes. They work by sorting records based on the predicted value and confidence of the prediction, splitting the records into groups of equal size (quantiles), and then plotting the value of the business criterion for each quantile, from highest to lowest. Multiple models are shown as separate lines in the plot.

<table>
<thead>
<tr>
<th>Strong Links</th>
<th>Field 1</th>
<th>Field 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>Cholesterol = &quot;HIGH&quot;</td>
<td>Drug = &quot;drugY&quot;</td>
</tr>
<tr>
<td>44</td>
<td>Cholesterol = &quot;NORMAL&quot;</td>
<td>Drug = &quot;drugY&quot;</td>
</tr>
<tr>
<td>42</td>
<td>BP = &quot;HIGH&quot;</td>
<td>Cholesterol = &quot;NORMAL&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Medium Links</th>
<th>Field 1</th>
<th>Field 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>BP = &quot;HIGH&quot;</td>
<td>Drug = &quot;drugY&quot;</td>
</tr>
<tr>
<td>37</td>
<td>BP = &quot;NORMAL&quot;</td>
<td>Cholesterol = &quot;HIGH&quot;</td>
</tr>
<tr>
<td>36</td>
<td>BP = &quot;NORMAL&quot;</td>
<td>Drug = &quot;drugX&quot;</td>
</tr>
<tr>
<td>35</td>
<td>BP = &quot;HIGH&quot;</td>
<td>Cholesterol = &quot;HIGH&quot;</td>
</tr>
<tr>
<td>34</td>
<td>Cholesterol = &quot;NORMAL&quot;</td>
<td>Drug = &quot;drugX&quot;</td>
</tr>
<tr>
<td>33</td>
<td>BP = &quot;LOW&quot;</td>
<td>Cholesterol = &quot;NORMAL&quot;</td>
</tr>
<tr>
<td>31</td>
<td>BP = &quot;LOW&quot;</td>
<td>Cholesterol = &quot;HIGH&quot;</td>
</tr>
<tr>
<td>30</td>
<td>BP = &quot;LOW&quot;</td>
<td>Drug = &quot;drugY&quot;</td>
</tr>
<tr>
<td>23</td>
<td>BP = &quot;NORMAL&quot;</td>
<td>Drug = &quot;drugX&quot;</td>
</tr>
<tr>
<td>22</td>
<td>BP = &quot;NORMAL&quot;</td>
<td>Cholesterol = &quot;NORMAL&quot;</td>
</tr>
<tr>
<td>20</td>
<td>Cholesterol = &quot;HIGH&quot;</td>
<td>Drug = &quot;drugX&quot;</td>
</tr>
<tr>
<td>18</td>
<td>BP = &quot;LOW&quot;</td>
<td>Drug = &quot;drugY&quot;</td>
</tr>
<tr>
<td>16</td>
<td>BP = &quot;LOW&quot;</td>
<td>Drug = &quot;drugX&quot;</td>
</tr>
<tr>
<td>15</td>
<td>Cholesterol = &quot;HIGH&quot;</td>
<td>Drug = &quot;drugC&quot;</td>
</tr>
<tr>
<td>16</td>
<td>BP = &quot;HIGH&quot;</td>
<td>Drug = &quot;drugB&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weak Links</th>
<th>Field 1</th>
<th>Field 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Cholesterol = &quot;HIGH&quot;</td>
<td>Drug = &quot;drugA&quot;</td>
</tr>
<tr>
<td>11</td>
<td>Cholesterol = &quot;NORMAL&quot;</td>
<td>Drug = &quot;drugA&quot;</td>
</tr>
<tr>
<td>8</td>
<td>Cholesterol = &quot;HIGH&quot;</td>
<td>Drug = &quot;drugB&quot;</td>
</tr>
<tr>
<td>8</td>
<td>Cholesterol = &quot;NORMAL&quot;</td>
<td>Drug = &quot;drugB&quot;</td>
</tr>
</tbody>
</table>
Outcomes are handled by defining a specific value or range of values as a **hit**. Hits usually indicate success of some sort (such as a sale to a customer) or an event of interest (such as a specific medical diagnosis). You can define hit criteria on the Options tab of the dialog box. Or, you can use the default hit criteria as follows:

- **Flag** output fields are straightforward; hits correspond to *true* values.
- For **Set** output fields, the first value in the set defines a hit.
- For **Range** output fields, hits equal values greater than the midpoint of the field's range.

There are five types of evaluation charts, each of which emphasizes a different evaluation criterion.

**Gains Charts**

Gains are defined as the proportion of total hits that occurs in each quantile. Gains are computed as \((\text{number of hits in quantile} / \text{total number of hits}) \times 100\%\).

*Figure 10-48*

*Gains chart (cumulative) with baseline, best line and business rule displayed*
Lift Charts

Lift compares the percentage of records in each quantile that are hits with the overall percentage of hits in the training data. It is computed as \((\text{hits in quantile} / \text{records in quantile}) / (\text{total hits} / \text{total records})\).

Figure 10-49
*Lift chart (cumulative) using points and split into even bands*

Response Charts

Response is simply the percentage of records in the quantile that are hits. Response is computed as \((\text{hits in quantile} / \text{records in quantile}) \times 100\%\).
**Profit Charts**

Profit equals the **revenue** for each record minus the **cost** for the record. Profits for a quantile are simply the sum of profits for all records in the quantile. Profits are assumed to apply only to hits, but costs apply to all records. Profits and costs can be fixed or can be defined by fields in the data. Profits are computed as (sum of revenue for records in quantile – sum of costs for records in quantile).
ROI (return on investment) is similar to profit in that it involves defining revenues and costs. ROI compares profits to costs for the quantile. ROI is computed as (profits for quantile/costs for quantile) × 100%.
Evaluation charts can be also be cumulative, so that each point equals the value for the corresponding quantile plus all higher quantiles. Cumulative charts usually convey the overall performance of models better, whereas non-cumulative charts often excel at indicating particular problem areas for models.
**Chapter 10**

**Setting Options for the Evaluation Chart Node**

Figure 10-53
Setting options for an Evaluation Chart node

![Diagram of Evaluation Chart Node Options]

- **Chart type.** Select one of following types: Gains, Response, Lift, Profit, or ROI (return on investment).

- **Cumulative plot.** Select to create a cumulative chart. Values in cumulative charts are plotted for each quantile plus all higher quantiles.

- **Include baseline.** Select to include a baseline in the plot, indicating a perfectly random distribution of hits where confidence becomes irrelevant. (Include baseline is not available for Profit and ROI charts.)

- **Include best line.** Select to include a best line in the plot, indicating perfect confidence (where hits = 100% of cases).

- **Plot.** Select the size of quantiles to plot in the chart from the drop-down list. Options include Quartiles, Quintiles, Deciles, Vingtiles, Percentiles, and 1000-tiles.
**Style.** Select Line or Point. Specify a **point type** by selecting one from the drop-down list. Options include Dot, Rectangle, Plus, Triangle, Hexagon, Horizontal dash, and Vertical dash.

For Profit and ROI charts, additional controls allow you to specify costs, revenue, and weights.

- **Costs.** Specify the cost associated with each record. You can select Fixed or Variable costs. For fixed costs, specify the cost value. For variable costs, click the Field Chooser button to select a field as the cost field.

- **Revenue.** Specify the revenue associated with each record that represents a hit. You can select Fixed or Variable costs. For fixed revenue, specify the revenue value. For variable revenue, click the Field Chooser button to select a field as the revenue field.

- **Weight.** If the records in your data represent more than one unit, you can use frequency weights to adjust the results. Specify the weight associated with each record, using Fixed or Variable weights. For fixed weights, specify the weight value (the number of units per record). For variable weights, click the Field Chooser button to select a field as the weight field.

### Setting Additional Options for Evaluation Charts

The Options tab for evaluation charts provides flexibility in defining hits, scoring criteria, and business rules displayed in the chart. You can also set options for exporting the results of the model evaluation.
Figure 10-54
Options tab settings for an Evaluation Chart node

User defined hit. Select to specify a custom condition used to indicate a hit. This option is useful for defining the outcome of interest rather than deducing it from the type of target field and the order of values.

Condition. When User defined hit is selected above, you must specify a CLEM expression for a hit condition. For example, @TARGET = “YES” is a valid condition indicating that a value of Yes for the target field will be counted as a hit in the evaluation. The specified condition will be used for all target fields. To create a condition, type in the field or use the Expression Builder to generate a condition expression. If the data are instantiated, you can insert values directly from the Expression Builder.
**User defined score.** Select to specify a condition used for scoring cases before assigning them to quantiles. The default score is calculated from the predicted value and the confidence. Use the Expression field below to create a custom scoring expression.

**Expression.** Specify a CLEM expression used for scoring. For example, if a numeric output in the range 0–1 is ordered so that lower values are better than higher, you might define a hit above as @TARGET < 0.5 and the associated score as 1 – @PREDICTED. The score expression must result in a numeric value. To create a condition, type in the field or use the Expression Builder to generate a condition expression.

**Include business rule.** Select to specify a rule condition reflecting criteria of interest. For example, you may want to display a rule for all cases where mortgage = "Y" and income >= 33000. Business rules are drawn on the chart as for predicted fields and labeled in the key as Rule.

**Condition.** Specify a CLEM expression used to define a business rule in the output chart. Simply type in the field or use the Expression Builder to generate a condition expression. If the data are instantiated, you can insert values directly from the Expression Builder.

**Export results to file.** Select to export the results of the model evaluation to a delimited text file. You can read this file to perform specialized analyses on the calculated values. Set the following options for export:

- **Filename.** Enter the filename for the output file. Use the ellipses button (...) to browse to the desired directory. The default directory is the current server or local directory.
- **Delimiter.** Enter a character, such as a comma or space, to use as the field delimiter.
- **Include field names.** Select this option to include field names as the first line of the output file.
- **New line after each record.** Select this option to begin each record on a new line.

**Reading the Results of a Model Evaluation**

The interpretation of an evaluation chart depends to a certain extent on the type of chart, but there are some characteristics common to all evaluation charts. For cumulative charts, higher lines indicate better models, especially on the left side of the
Chapter 10

chart. In many cases, when comparing multiple models the lines will cross, so that one model will be higher in one part of the chart and another will be higher in a different part of the chart. In this case, you need to consider what portion of the sample you want (which defines a point on the x axis) when deciding which model to choose.

Most of the non-cumulative charts will be very similar. For good models, non-cumulative charts should be high toward the left side of the chart and low toward the right side of the chart. (If a non-cumulative chart shows a sawtooth pattern, you can smooth it out by reducing the number of quantiles to plot and reexecuting the graph.) Dips on the left side of the chart or spikes on the right side can indicate areas where the model is predicting poorly. A flat line across the whole graph indicates a model that essentially provides no information.

**Gains charts.** Cumulative gains charts always start at 0% and end at 100% as you go from left to right. For a good model, the gains chart will rise steeply toward 100% and then level off. A model that provides no information will follow the diagonal from lower left to upper right (shown in the chart if Include baseline is selected).

**Lift charts.** Cumulative lift charts tend to start above 1.0 and gradually descend until they reach 1.0 as you go from left to right. The right edge of the chart represents the entire data set, so the ratio of hits in cumulative quantiles to hits in data is 1.0. For a good model, lift should start well above 1.0 on the left, remain on a high plateau as you move to the right, and then trail off sharply toward 1.0 on the right side of the chart. For a model that provides no information, the line will hover around 1.0 for the entire graph. (If Include baseline is selected, a horizontal line at 1.0 is shown in the chart for reference.)

**Response charts.** Cumulative response charts tend to be very similar to lift charts except for the scaling. Response charts usually start near 100% and gradually descend until they reach the overall response rate (total hits / total records) on the right edge of the chart. For a good model, the line will start near or at 100% on the left, remain on a high plateau as you move to the right, and then trail off sharply toward the overall response rate on the right side of the chart. For a model that provides no information, the line will hover around the overall response rate for the entire graph. (If Include baseline is selected, a horizontal line at the overall response rate is shown in the chart for reference.)

**Profit charts.** Cumulative profit charts show the sum of profits as you increase the size of the selected sample, moving from left to right. Profit charts usually start near zero, increase steadily as you move to the right until they reach a peak or plateau in
the middle, and then decrease toward the right edge of the chart. For a good model, profits will show a well-defined peak somewhere in the middle of the chart. For a model that provides no information, the line will be relatively straight and may be increasing, decreasing, or level depending on the cost/revenue structure that applies.

**ROI charts.** Cumulative ROI (return on investment) charts tend to be similar to response charts and lift charts except for the scaling. ROI charts usually start above 0% and gradually descend until they reach the overall ROI for the entire data set (which can be negative). For a good model, the line should start well above 0%, remain on a high plateau as you move to the right, and then trail off rather sharply toward the overall ROI on the right side of the chart. For a model that provides no information, the line should hover around the overall ROI value.

**Using an Evaluation Chart**

Using the mouse to explore an evaluation chart is similar to using a histogram or collection graph.
The $x$ axis represents model scores across the specified quantiles, such as vingtiles or deciles. You can partition the $x$ axis into bands just as you would for a histogram by clicking with the mouse or using the splitter icon to display options for automatically splitting the axis into equal bands.

You can manually edit the boundaries of bands by selecting Graph Bands from the Edit menu. For more information, see “Editing Graph Bands” on page 271.
Using Bands to Produce Feedback

Once you have a defined a band, there are numerous ways to delve deeper into the
selected area of the graph. Use the mouse in the following ways to produce feedback
in the graph window:

- Hover over bands to provide point-specific information.
- Check the range for a band by right-clicking inside a band and reading the
  feedback panel at the bottom of the window.
- Right-click in a band to bring up a context menu with additional options, such
  as generating process nodes.
- Rename bands by right-clicking in a band and selecting Rename Band. By
default, bands are named bandN, where N equals the number of bands from left
to right on the x axis.
- Move the boundaries of a band by selecting a band line with your mouse and
  moving it to the desired location on the x axis.
- Delete bands by right-clicking on a line and selecting Delete Band.

Generating Nodes

Once you have created an evaluation chart, defined bands, and examined the results,
you can use options on the Generate menu and the context menu to automatically
create nodes based upon selections in the graph.

- Generate a Select or Derive Flag node based on inclusion in a particular band's
  range of values.
- Generate a Derive Set node to indicate which band contains the record based upon
  score and hit criteria for the model.

Selecting a Model

When generating nodes from an Evaluation Chart, you will be prompted to select a
single model from all available in the chart.
Select a model and click OK to generate the new node onto the stream canvas.

**Hiding and Showing Lines**

To hide or show lines in the evaluation chart output:

- Right-click on the legend.

- From the context menu, select specific lines to hide or show. Select Hide All or Show All to perform the action for all available lines.
Alternatively, you can select options from the Edit menu on the output browser. Select Hide Lines to open a separate dialog box where you can view all lines for each target field.

**Figure 10-59**

*Hide/Show Lines dialog box*

Use the Visible column to determine whether the line appears on the chart.
Chapter 11

Modeling Nodes

Overview of Modeling Nodes

Modeling nodes are the heart of the data mining process. The methods available in these nodes allow you to derive new information from your data and develop predictive models. Clementine offers a variety of modeling methods taken from machine learning, artificial intelligence, and statistics. Each method has certain strengths and is best suited for particular types of problems.

Figure 11-1
Modeling palette

The Modeling palette contains the following nodes:

- Neural Net
- C5.0
- Kohonen
- Linear Regression
- Generalized Rule Induction (GRI)
- Apriori
- K-Means
- Logistic Regression
- Factor Analysis/PCA
- TwoStep Cluster
Chapter 11

- Classification and Regression (C&R) Trees
- Sequence Detection

The following topics provide information on using Modeling nodes in Clementine as well as an introduction to modeling methods. For an in-depth understanding of the algorithms used in Clementine, consult the *Clementine Algorithms Guide* available on the product CD.

**Modeling Node Fields Options**

All modeling nodes have a Fields tab, where you can specify the fields to be used in building the model.

**Figure 11-2**
*Fields tab for Neural Net node*
Before you can build a model, you need to specify which fields you want to use as targets and as inputs. By default, all modeling nodes except the Sequence node will use field information from an upstream Type node. If you are using a Type node to select input and target fields, you don't need to change anything on this tab. For Sequence models, you must specify the field settings on the Fields tab of the modeling node. For more information, see “Sequence Node Field Options” on page 374.

**Use Type node settings.** This option tells the node to use field information from an upstream type node. This is the default.

**Use custom settings.** This option tells the node to use field information specified here instead of that given in any upstream Type node(s). After selecting this option, specify fields below.

- **Target(s).** For models that require one or more target fields, select the target field(s). This is similar to setting a field's direction to Out in a Type node.

- **Inputs.** Select the input field(s). This is similar to setting a field's direction to In in a Type node.

- **Use frequency field.** This option allows you to select a field as a frequency weight. Use this if the records in your training data represent more than one unit each—for example, if you are using aggregated data. The field values should be the number of units represented by each record. Values for a frequency field should be positive integers. Frequency weights affect calculation of branch instances for C&RT models. Records with negative or zero frequency weight are excluded from the analysis. Non-integer frequency weights are rounded to the nearest integer.

- **Use weight field.** This option allows you to select a field as a case weight. Case weights are used to account for differences in variance across levels of the output field. These weights are used in model estimation but do not affect calculation of branch instances for C&RT models. Case weight values should be positive but need not be integer values. Records with negative or zero case weight are excluded from the analysis.
Chapter 11

- **Consequents.** For rule induction nodes (Apriori and GRI), select the fields to be used as consequents in the resulting ruleset. (This corresponds to fields with type *Out* or *Both* in a Type node.)

- **Antecedents.** For rule induction nodes (Apriori and GRI), select the fields to be used as antecedents in the resulting ruleset. (This corresponds to fields with type *In* or *Both* in a Type node.)

**Transactional data format (Apriori node only).** Apriori can handle data in either of two formats. **Transactional** data has two fields: one for an ID and one for content. Each record represents a single item, and associated items are linked by having the same ID. Here is an example of data in **Transactional** format:

<table>
<thead>
<tr>
<th>Customer</th>
<th>Purchase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>jam</td>
</tr>
<tr>
<td>2</td>
<td>milk</td>
</tr>
<tr>
<td>3</td>
<td>jam</td>
</tr>
<tr>
<td>3</td>
<td>bread</td>
</tr>
<tr>
<td>4</td>
<td>jam</td>
</tr>
<tr>
<td>4</td>
<td>bread</td>
</tr>
<tr>
<td>4</td>
<td>milk</td>
</tr>
</tbody>
</table>

**Tabular data format (Apriori node only).** Tabular data has items represented by separate flags, and each record represents a complete set of associated items. Here is an example of **Tabular** data:

<table>
<thead>
<tr>
<th>Customer</th>
<th>jam</th>
<th>bread</th>
<th>milk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Neural Net Node

The Neural Net node (formerly called “Train Net”) is used to create and train a neural network. A **neural network**, sometimes called a **multi-layer perceptron**, is basically a simplified model of the way the human brain processes information. It works by simulating a large number of interconnected simple processing units that resemble abstract versions of **neurons**.

The processing units are arranged in layers. There are typically three parts in a neural network: an **input layer** with units representing the input fields, one or more **hidden layers**, and an **output layer** with a unit or units representing the output field(s). The units are connected with varying connection strengths or **weights**.

The network learns by examining individual records, generating a prediction for each record, and making adjustments to the weights whenever it makes an incorrect prediction. This process is repeated many times, and the network continues to improve its predictions until one or more of the stopping criteria have been met.

**Requirements.** There are no restrictions on field types. Neural Net nodes can handle numeric, symbolic, or flag inputs and outputs. The Neural Net node expects one or more fields with direction **In** and one or more fields with direction **Out**. Fields set to **Both** or **None** are ignored. Field types must be fully instantiated when the node is executed.

**Strengths.** Neural networks are powerful general function estimators. They usually perform prediction tasks at least as well as other techniques and sometimes perform significantly better. They also require minimal statistical or mathematical knowledge to train or apply. Clementine incorporates several features to avoid some of the common pitfalls of neural networks, including **sensitivity analysis** to aid in interpretation of the network, **pruning** and **validation** to prevent overtraining, and **dynamic** networks to automatically find an appropriate network architecture.
Neural Net Node Model Options

Editing the Neural Net node allows you to set the parameters for the node. You can set the following parameters:

Model name. Specify the name of the network to be produced.

- Auto. With this option selected, the model name will be generated automatically, based on the target field name(s). This is the default.
- Custom. Select this option to specify your own name for the generated model that will be created by this node.

Method. Clementine provides six training methods for building neural network models:

- Quick. This method uses rules of thumb and characteristics of the data to choose an appropriate shape (topology) for the network. Note that the method for calculating default size of the hidden layer has changed from previous versions of
Clementine. The new method will generally produce smaller hidden layers that are faster to train and that generalize better. If you find that you get poor accuracy with the default size, try increasing the size of the hidden layer on the Expert tab or try an alternate training method.

- **Dynamic.** This method creates an initial topology, but modifies the topology by adding and/or removing hidden units as training progresses.

- **Multiple.** This method creates several networks of different topologies (the exact number depends on the training data). These networks are then trained in a pseudo-parallel fashion. At the end of training, the model with the lowest RMS error is presented as the final model.

- **Prune.** This method starts with a large network and removes (prunes) the weakest units in the hidden and input layers as training proceeds. This method is usually slow, but it often yields better results than other methods.

- **RBFN.** The radial basis function network (RBFN) uses a technique similar to \( k \)-means clustering to partition the data based on values of the target field.

- **Exhaustive prune.** This method is related to the Prune method. It starts with a large network and prunes the weakest units in the hidden and input layers as training proceeds. With Exhaustive Prune, network training parameters are chosen to ensure a very thorough search of the space of possible models to find the best one. This method is usually the slowest, but it often yields the best results. Note that this method can take a long time to train, especially with large data sets.

**Prevent overtraining.** When selected, this option splits the data randomly into training and validation sets. The network is trained on the training set, and accuracy is estimated based on the validation set. You can specify the proportion of the data to be used for training in the Sample % box. (The remainder of the data is used for validation.)

**Set random seed.** If no random seed is set, the sequence of random values used to initialize the network weights will be different every time the node is executed. This can cause the node to create different models on different runs, even if the network settings and data values are exactly the same. By selecting this option, you can set the random seed to a specific value so that the resulting model is exactly reproducible. A specific random seed always generates the same sequence of random values, in which case executing the node always yields the same generated model.
Stop on. You can select one of the following stopping criteria:

- **Default.** With this setting, the network will stop training when the network appears to have reached its optimally trained state. If the default setting is used with the Multiple training method, the networks that fail to train well are discarded as training progresses.

- **Accuracy (%).** With this option, training will continue until the specified accuracy is attained. This may never happen, but you can interrupt training at any point and save the net with the best accuracy achieved so far.

- **Cycles.** With this option, training will continue for the specified number of cycles (passes through the data).

- **Time (mins).** With this option, training will continue for the specified amount of time (in minutes). Note that training may go a bit beyond the specified time limit in order to complete the current cycle.
Neural Net Node Additional Options

Figure 11-4
Neural Net node options

Continue training existing model. By default, each time you execute a Neural Net node, a completely new network is created. If you select this option, training continues with the last net successfully produced by the node. The node correctly handles changes of training method between runs, except that RBFN networks cannot be adapted to other types of networks. Thus, when changing to or from the RBFN method, a new network is always created when the changed node is executed.

Use binary set encoding. If this option is selected, Clementine will use a compressed binary encoding scheme for set fields. This option allows you to more easily build neural net models using set fields with large numbers of values as inputs. However, if you use this option, you may need to increase the complexity of the network architecture (by adding more hidden units or more hidden layers) to allow the network to properly use the compressed information in binary encoded set fields.
**Show feedback graph.** If this option is selected, you will see a graph that displays the accuracy of the network over time as it learns. In addition, if you have selected Generate log file, you will see a second graph showing the training set and test set metrics (defined below).

*Note:* This feature can slow training time. To speed training time, deselect this option.

![Neural Net feedback graph](image)

**Model selection.** By default, when training is interrupted, the node will return the Best network as the generated net node. You can request that the node return the Final model instead.

**Sensitivity analysis.** With this option selected, a sensitivity analysis of the input fields will be performed after the network has been trained. The sensitivity analysis provides information on which input fields are most important in predicting the output field(s). (These results are part of the model information available in the generated model browser.)

**Generate log file.** If this option is selected, information on training progress will be written to the specified log file. To change the log file, enter a Log filename or use the File Chooser button to select a location. (If you select a file that already exists, the new information will be appended to the file.)

The format of each entry in the log file is as follows:

```
<Time> <Net ID> <Training Cycle> <Training Set Metric> <Test Set Metric>
```

- **<Time>** takes the format HH:MM:SS.
- **<Net ID>** indicates which network is being trained when the network is in Multiple training mode. For other training modes, the value is always 1.
- **<Training Cycle>** is an integer, incrementing from 0 on each training run.
- **<Training Set Metric>** and **<Test Set Metric>** are measures of network performance on the training data and test data, respectively. (These values are identical when Prevent overtraining is deselected.) They are calculated as the squared correlation between predicted and actual values divided by the mean squared error (MSE). If both Generate log file and Show feedback graph are selected, these metrics are displayed in the feedback graph in addition to the usual accuracy values.

**Neural Net Node Expert Options—Quick Method**

**Figure 11-6**
*Quick method expert options*

- **Hidden layers.** Select the number of hidden layers for the neural network. More hidden layers can help neural networks learn more complex relationships, but they also increase training time.
Layer 1, 2, 3. For each layer, specify the number of hidden units to include. More hidden units per layer can also help in learning complex tasks, but as with additional hidden layers, they also increase training time.

Persistence. Specify the number of cycles for which the network will continue to train if no improvement is seen. Higher values can help networks escape local minima, but they also increase training time.

Alpha and Eta. These parameters control the training of the network. For more information, see “Neural Net Node Learning Rates” on page 323.

**Neural Net Node Expert Options—Dynamic Method**

There are no expert options for the dynamic method in the Neural Net node.
Neural Net Node Expert Options - Multiple Method

Topologies. Specify the topologies of the networks to be trained. A topology is given by specifying the number of hidden units in each layer, separated by commas. Topologies can specify one, two, or three hidden layers by using the appropriate number of parameters. For example, a network with one hidden layer of 10 units would be specified as 10; a network with three hidden layers of 10, 12, and 15 units would be specified as 10, 12, 15.

You can also specify a range of numbers for hidden units in a layer by providing two or three numbers separated by spaces. If two numbers are given, separate networks are created with a number of hidden units equal to each integer between the first and second number (inclusive). For example, to generate networks having 10, 11, 12, 13, and 14 hidden units in a single layer, specify 10 14. To generate networks with two hidden layers where the first layer varies from 10 to 14 and the second layer varies from 8 to 12, specify 10 14, 8 12. In this case, networks are generated that contain all possible combinations of values. If a third value is given, it is used as an...
increment for counting from the first value to the second. For example, to generate networks with 10, 12, 14, and 16 hidden units, specify 10 16 2.

Finally, you can provide multiple network topologies, separated by semicolons. For example, to generate networks with one hidden layer of 10, 12, 14, and 16 hidden units, and networks having two hidden layers of 10 hidden units and 7 to 10 hidden units, respectively, specify 10 16 2; 10, 7 10.

Discard non-pyramids. Pyramids are networks where each layer contains the same number or fewer hidden units than the preceding layer. Such networks usually train better than non-pyramidal networks. Selecting this option discards networks that are not pyramids.

Persistence. Specify the number of cycles for which the network will continue to train if no improvement is seen.

Alpha and Eta. These parameters control the training of the network. For more information, see “Neural Net Node Learning Rates” on page 323.
Neural Net Node Expert Options—Prune Method

Hidden layers. Select the number of hidden layers for the initial network (before pruning).

Layer 1, 2, 3. For each layer, specify the number of hidden units to include in the initial network (before pruning). The initial layers should be slightly larger than you would use with another training method.

Hidden rate. Specify the number of hidden units to be removed in a single hidden unit pruning.

Hidden persistence. Specify the number of hidden unit pruning operations to perform if no improvement is seen.

Input rate. Specify the number of input units to be removed in a single input pruning.
**Input persistence.** Specify the number of input pruning operations to be performed if no improvement is seen.

**Persistence.** Specify the number of cycles for which the network will train before attempting to prune if no improvement is seen.

**Overall persistence.** Specify the number of times to go through the hidden unit prune/input prune loop if no improvement is seen. Applies when using the Default stopping model. For more information, see “Neural Net Node Model Options” on page 312.

**Alpha and Eta.** These parameters control the training of the network. For more information, see “Neural Net Node Learning Rates” on page 323.

**Neural Net Node Expert Options—RBFN Method**

*Figure 11-9*

*RBFN method expert options*
**RBF clusters.** Specify the number of radial basis functions or clusters to use. This corresponds to the size of the hidden layer.

**Persistence.** Specify the number of cycles for which the network will continue to train if no improvement is seen.

**Eta.** For RBFNs, eta remains constant. By default, eta will be computed automatically, based on the first two cycles. To specify the value for eta, deselect Compute Eta automatically and enter the desired value. For more information, see “Neural Net Node Learning Rates” on page 323.

**Alpha.** A momentum term used in updating the weights during training. For more information, see “Neural Net Node Learning Rates” on page 323.

**RBF overlapping.** The hidden units in an RBFN represent radial basis functions that define clusters or regions in the data. This parameter allows you to control how much those regions or clusters overlap. Normally during training, records affect only the cluster(s) to which they are closest. By increasing this parameter, you increase the size of the region associated with each hidden unit, allowing records to affect more distant clusters. Specify a positive real value.

**Neural Net Node Expert Options—Exhaustive Prune Method**

There are no expert options for the Exhaustive Prune method in the Neural Net node.

**Neural Net Node Learning Rates**

Neural net training is controlled by several parameters. These parameters can be set using the Expert tab of the Neural Net node dialog box.

**Alpha.** A momentum term used in updating the weights during training. **Momentum** tends to keep the weight changes moving in a consistent direction. Specify a value between 0 and 1. Higher values of alpha can help the network escape from local minima.

**Eta.** The learning rate, which controls how much the weights are adjusted at each update. Eta changes as training proceeds for all training methods except RBFN, where eta remains constant. Initial Eta is the starting value of Eta. During training, Eta starts at Initial Eta, decreases to Low Eta, then is reset to High Eta and decreases
to Low Eta again. The last two steps are repeated until training is complete. This process is shown in the following figure.

**Figure 11-10**
*How eta changes during neural network training*

Initial Eta → ![Graph showing how eta changes during training](image)

High Eta → Low Eta → Cycles

Eta decay specifies the rate at which eta decreases, expressed as the number of cycles to go from High Eta to Low Eta. Specify values for each Eta option.

**Kohonen Node**

The Kohonen node is used to create and train a special type of neural network called a **Kohonen network**, a **knet**, or a **self-organizing map**. This type of network can be used to cluster the data set into distinct groups, when you don't know what those groups are at the beginning. Unlike most learning methods in Clementine, Kohonen networks do **not** use a target field. This type of learning, with no target field, is called **unsupervised learning**. Instead of trying to predict an outcome, Kohonen nets try to uncover patterns in the set of input fields. Records are grouped so that records within a group or cluster tend to be similar to each other, and records in different groups are dissimilar.

A Kohonen network consists of an input layer of units and a two-dimensional output grid of processing units. During training, each unit competes with all of the others to “win” each record. When a unit wins a record, its weights (along with those of other nearby units, collectively referred to as a **neighborhood**) are adjusted to better match the pattern of predictor values for that record. As training proceeds, the weights on the grid units are adjusted so that they form a two-dimensional “map”...
of the clusters. (Hence the term **self-organizing map**.) Usually, a Kohonen net will end up with a few units that summarize many observations (**strong** units), and several units that don’t really correspond to any of the observations (**weak** units). The strong units (and sometimes other units adjacent to them in the grid) represent probable cluster centers.

Another use of Kohonen networks is in **dimension reduction**. The spatial characteristic of the two-dimensional grid provides a mapping from the \( k \) original predictors to two derived features that preserve the similarity relationships in the original predictors. In some cases, this can give you the same kind of benefit as factor analysis or PCA.

Note that the method for calculating default size of the output grid has changed from previous versions of Clementine. The new method will generally produce smaller output layers that are faster to train and generalize better. If you find that you get poor results with the default size, try increasing the size of the output grid on the Expert tab. For more information, see “Kohonen Node Expert Options” on page 328.

**Requirements.** To train a Kohonen net, you need one or more **In** fields. Fields set as **Out**, **Both**, or **None** are ignored.

**Strengths.** You do not need to have data on group membership to build a Kohonen network model. You don’t even need to know the number of groups to look for. Kohonen networks start with a large number of units, and as training progresses, the units gravitate toward the natural clusters in the data. You can look at the number of observations captured by each unit in the generated model to identify the strong units, which can give you a sense of the appropriate number of clusters.
Kohonen Node Model Options

Figure 11-11
Kohonen node model options

Model name. Specify the name of the network to be produced.

- **Auto.** With this option selected, the model name will be “Kohonen.” This is the default.
- **Custom.** Select this option to specify your own name for the generated model that will be created by this node.

Continue training existing model. By default, each time you execute a Kohonen node, a completely new network is created. If you select this option, training continues with the last net successfully produced by the node.

Show feedback graph. If this option is selected, a visual representation of the two-dimensional array is displayed during training. The strength of each node is represented by color. Red denotes a unit that is winning many records (a strong unit), and white denotes a unit that is winning few or no records (a weak unit). Note that this feature can slow training time. To speed up training time, deselect this option.
Stop on. The default stopping criterion stops training, based on internal parameters. You can also specify time as the stopping criterion. Enter the time (in minutes) for the network to train.

Set random seed. If no random seed is set, the sequence of random values used to initialize the network weights will be different every time the node is executed. This can cause the node to create different models on different runs, even if the node settings and data values are exactly the same. By selecting this option, you can set the random seed to a specific value so that the resulting model is exactly reproducible. A specific random seed always generates the same sequence of random values, in which case executing the node always yields the same generated model.

Note: Use binary set encoding, an option available in previous versions of Clementine, has been removed. In some situations, that option tended to distort distance information between records and thus was not suitable for use with Kohonen nets, which rely heavily on such distance information. If you want to include set fields in your model but are having memory problems in building the model, or the model is taking too long to build, consider recoding large set fields to reduce the number of values or using a different field with fewer values as a proxy for the large set. For example, if you are having a problem with a product_id field containing values for individual products, you might consider removing it from the model and adding a less detailed product_category field instead.
Kohonen Node Expert Options

For those with detailed knowledge of Kohonen networks, expert options allow you to fine-tune the training process. To access expert options, set the Mode to Expert on the Expert tab.

Figure 11-13
Kohonen expert options

**Width and Length.** Specify the size (width and length) of the two-dimensional output map as number of output units along each dimension.

**Learning rate decay.** Select either linear or exponential learning rate decay. The learning rate is a weighting factor that decreases over time, such that the network starts off encoding large-scale features of the data and gradually focuses on more fine-level detail.

**Phase 1 and Phase 2.** Kohonen net training is split into two phases. Phase 1 is a rough estimation phase, used to capture the gross patterns in the data. Phase 2 is a tuning phase, used to adjust the map to model the finer features of the data. For each phase, there are three parameters:

- **Neighborhood.** Sets the starting size (radius) of the neighborhood. This determines the number of “nearby” units that get updated along with the winning unit during training. During phase 1, the neighborhood size starts at Phase 1 Neighborhood...
and decreases to \((\text{Phase 2 Neighborhood} + 1)\). During phase 2, neighborhood size starts at \(\text{Phase 2 Neighborhood}\) and decreases to 1.0. \(\text{Phase 1 Neighborhood}\) should be larger than \(\text{Phase 2 Neighborhood}\).

- **Initial Eta.** Sets the starting value for learning rate \(\text{eta}\). During phase 1, \(\text{eta}\) starts at \(\text{Phase 1 Initial Eta}\) and decreases to \(\text{Phase 2 Initial Eta}\). During phase 2, \(\text{eta}\) starts at \(\text{Phase 2 Initial Eta}\) and decreases to 0. \(\text{Phase 1 Initial Eta}\) should be larger than \(\text{Phase 2 Initial Eta}\).

- **Cycles.** Sets the number of cycles for each phase of training. Each phase continues for the specified number of passes through the data.

**C5.0 Node**

This node uses the C5.0 algorithm to build either a **decision tree** or a **ruleset**. A C5.0 model works by splitting the sample based on the field that provides the maximum **information gain**. Each subsample defined by the first split is then split again, usually based on a different field, and the process repeats until the subsamples cannot be split any further. Finally, the lowest level splits are reexamined, and those that do not contribute significantly to the value of the model are removed or **pruned**.

C5.0 can produce two kinds of models. A **decision tree** is a straightforward description of the splits found by the algorithm. Each terminal, or “leaf,” node describes a particular subset of the training data, and each case in the training data belongs to exactly one terminal node in the tree. In other words, exactly one prediction is possible for any particular data record presented to a decision tree.

In contrast, a **ruleset** is a set of rules that tries to make predictions for individual records. Rulesets are derived from decision trees and, in a way, represent a simplified or distilled version of the information found in the decision tree. Rulesets can often retain most of the important information from a full decision tree but with a less complex model. Because of the way rulesets work, they do not have the same properties as decision trees. The most important difference is that with a ruleset, more than one rule may apply for any particular record, or no rules at all may apply. If multiple rules apply, each rule gets a weighted “vote” based on the confidence associated with that rule, and the final prediction is decided by combining the weighted votes of all of the rules that apply to the record in question. If no rule applies, a default prediction is assigned to the record.
**Requirements.** To train a C5.0 model, you need one or more *In* fields and one or more symbolic *Out* field(s). Fields set to *Both* or *None* are ignored. Fields used in the model must have their types fully instantiated.

**Strengths.** C5.0 models are quite robust in the presence of problems such as missing data and large numbers of input fields. They usually do not require long training times to estimate. In addition, C5.0 models tend to be easier to understand than some other model types, since the rules derived from the model have a very straightforward interpretation. C5.0 also offers the powerful **boosting** method to increase accuracy of classification.

### C5.0 Node Model Options

**Figure 11-14**  
**C5.0 node options**

- **Model name.** Specify the name of the model to be produced.
  - **Auto.** With this option selected, the model name will be generated automatically, based on the target field name(s). This is the default.
  - **Custom.** Select this option to specify your own name for the generated model that will be created by this node.
Output type. Specify here whether you want the resulting generated model to be a Decision tree or a Ruleset.

Group symbolics. If this option is selected, C5.0 will attempt to combine symbolic values that have similar patterns with respect to the output field. If this option is not selected, C5.0 will create a child node for every value of the symbolic field used to split the parent node. For example, if C5.0 splits on a COLOR field (with values RED, GREEN, and BLUE), it will create a three-way split by default. However, if this option is selected, and the records where COLOR = RED are very similar to records where COLOR = BLUE, it will create a two-way split, with the GREENs in one group and the BLUEs and REDs together in the other.

Use boosting. The C5.0 algorithm has a special method for improving its accuracy rate, called boosting. It works by building multiple models in a sequence. The first model is built in the usual way. Then, a second model is built such in such a way that it focuses especially on the records that were misclassified by the first model. Then a third model is built to focus on the second model's errors, and so on. Finally, cases are classified by applying the whole set of models to them, using a weighted voting procedure to combine the separate predictions into one overall prediction. Boosting can significantly improve the accuracy of a C5.0 model, but it also requires longer training. The Number of trials option allows you to control how many models are used for the boosted model. This feature is based on the research of Freund & Schapire, with some proprietary improvements to handle noisy data better.

Cross-validate. If this option is selected, C5.0 will use a set of models built on subsets of the training data to estimate the accuracy of a model built on the full data set. This is useful if your data set is too small to split into traditional training and testing sets. The cross-validation models are discarded after the accuracy estimate is calculated. You can specify the number of folds, or the number of models used for cross-validation. Note that in previous versions of Clementine, building the model and cross-validating it were two separate operations. In the current version, no separate model-building step is required. Model building and cross-validation are performed at the same time.

Mode. For Simple training, most of the C5.0 parameters are set automatically. Expert training allows more direct control over the training parameters.
Simple Mode Options

Favor. By default, C5.0 will try to produce the most accurate tree possible. In some instances, this can lead to overfitting, which can result in poor performance when the model is applied to new data. Select Generality to use algorithm settings that are less susceptible to this problem.

Note: Models built with the Generality option selected are not guaranteed to generalize better than other models. When generality is a critical issue, always validate your model against a held-out test sample.

Expected noise (%). Specify the expected proportion of noisy or erroneous data in the training set.

Expert Mode Options

Pruning severity. Determines the extent to which the generated decision tree or ruleset will be pruned. Increase this value to obtain a smaller, more concise tree. Decrease it to obtain a more accurate tree. This setting affects local pruning only (see “Use global pruning” below).

Minimum records per child branch. The size of subgroups can be used to limit the number of splits in any branch of the tree. A branch of the tree will be split only if two or more of the resulting subbranches would contain at least this many records from the training set. The default value is 2. Increase this value to help prevent overtraining with noisy data.

Use global pruning. Trees are pruned in two stages: First a local pruning stage which examines subtrees and collapses branches to increase the accuracy of the model. Second, a global pruning stage considers the tree as a whole, and weak subtrees may be collapsed. Global pruning is performed by default. To omit the global pruning stage, deselect this option.

Winnow attributes. If this option is selected, C5.0 will examine the usefulness of the predictors before starting to build the model. Predictors that are found to be irrelevant are then excluded from the model-building process. This option can be helpful for models with many predictor fields, and can help prevent overfitting.
**Misclassification Cost Options**

In some contexts, certain kinds of errors are more costly than others. For example, it may be more costly to classify a high-risk credit applicant as low risk (one kind of error) than it is to classify a low-risk applicant as high risk (a different kind). Misclassification costs allow you to specify the relative importance of different kinds of prediction errors.

**Figure 11-15**
*Specifying misclassification costs*

The misclassification cost matrix shows the cost for each possible combination of predicted category and actual category. By default, all misclassification costs are set to 1.0. To enter custom cost values, select *Use misclassification costs* and enter your custom values into the cost matrix.

To change a misclassification cost, select the cell corresponding to the desired combination of predicted and actual values, delete the existing contents of the cell, and enter the desired cost for the cell.

Remember that customized misclassification costs are not automatically symmetric. For example, if you set the cost of misclassifying $A$ as $B$ to be 2.0, the cost of misclassifying $B$ as $A$ will still have the default value of 1.0 unless you explicitly change it as well.
Chapter 11

**Linear Regression Node**

The Regression node generates a **linear regression** model. This model estimates the best fitting linear equation for predicting the output field, based on the input fields. The regression equation represents a straight line or plane that minimizes the squared differences between predicted and actual output values. This is a very common statistical technique for summarizing data and making predictions.

**Requirements.** Only numeric fields can be used in a regression model. You must have exactly one *Out* field and one or more *In* fields. Fields with direction *Both* or *None* are ignored, as are non-numeric fields.

**Strengths.** Regression models are relatively simple and give an easily interpreted mathematical formula for generating predictions. Because regression modeling is a long-established statistical procedure, the properties of these models are well understood. Regression models are also typically very fast to train. The Linear Regression node provides methods for automatic field selection in order to eliminate non-significant input fields from the equation.

**Linear Regression Node Model Options**

Figure 11-16
*Linear Regression node options*

- **Model name.** Specify the name of the model to be produced.
- **Auto.** With this option selected, the model name will be generated automatically, based on the target field name. This is the default.

- **Custom.** Select this option to specify your own name for the generated model that will be created by this node.

**Method.** Specify the method to be used in building the regression model.

- **Enter.** This is the default method, which enters all the In fields into the equation directly. No field selection is performed in building the model.

- **Stepwise.** The Stepwise method of field selection builds the equation in steps, as the name implies. The initial model is the simplest model possible, with no input fields in the equation. At each step, input fields that have not yet been added to the model are evaluated, and if the best of those input fields adds significantly to the predictive power of the model, it is added. In addition, input fields that are currently in the model are reevaluated to determine if any of them can be removed without significantly detracting from the model. If so, they are removed. Then the process repeats, and other fields are added and/or removed. When no more fields can be added to improve the model, and no more can be removed without detracting from the model, the final model is generated.

- **Backwards.** The Backwards method of field selection is similar to the Stepwise method in that the model is built in steps. However, with this method, the initial model contains all of the input fields as predictors, and fields can only be removed from the model. Input fields that contribute little to the model are removed one by one until no more fields can be removed without significantly worsening the model, yielding the final model.

- **Forwards.** The Forwards method is essentially the opposite of the Backwards method. With this method, the initial model is the simplest model with no input fields, and fields can only be added to the model. At each step, input fields not yet in the model are tested based on how much they would improve the model, and the best of those is added to the model. When no more fields can be added or the best candidate field does not produce a large enough improvement in the model, the final model is generated.

*Note:* The automatic methods (including Stepwise, Forwards, and Backwards) are highly adaptable learning methods and have a strong tendency to overfit the training data. When using these methods, it is especially important to verify the validity of the resulting model with a hold-out test sample or new data.
**Include constant in equation.** This option determines whether the resulting equation will include a constant term. In most situations, you should leave this selected. This option can be useful if you have prior knowledge that the output field equals 0 whenever the predictor field or fields equal 0.

**Linear Regression Node Expert Options**

For those with detailed knowledge of Linear Regression models, expert options allow you to fine-tune the model-building process. To access expert options, set the Mode to Expert on the Expert tab.

**Figure 11-17**

*Linear Regression expert options*

**Missing values.** By default, the Linear Regression node will use only records that have valid values for all fields used in the model. (This is sometimes called **listwise deletion** of missing values). If you have a lot of missing data, you may find that this approach eliminates too many records, leaving you without enough data to generate a good model. In such cases, you can deselect the **Only use complete records** option. Clementine will then attempt to use as much information as possible to estimate the Regression model, including records where some of the fields have missing values. (This is sometimes called **pairwise deletion** of missing values.) However, in some situations, using incomplete records in this manner can lead to computational problems in estimating the regression equation.
Singularity tolerance. This option allows you to specify the minimum proportion of variance in a field that must be independent of other fields in the model.

Stepping. These options allow you to control the criteria for adding and removing fields with the Stepwise, Forwards, or Backwards estimation methods. (The button is disabled if the Enter method is selected.) For more information, see “Linear Regression Node Stepping Options” on page 337.

Output. These options allow you to request additional statistics that will appear in the advanced output of the generated model built by the node. For more information, see “Linear Regression Node Output Options” on page 338.

**Linear Regression Node Stepping Options**

Figure 11-18
Linear Regression stepping options

Select one of the two criteria for stepping, and change the cut-off values as desired.

*Note:* There is an inverse relationship between the two criteria. The more important a field is for the model, the smaller the $p$ value, but the larger the $F$ value.

- **Use probability of F.** This option allows you to specify selection criteria based on the statistical probability (the *p value*) associated with each field. Fields will be added to the model only if the associated $p$ value is smaller than the Entry value and will be removed only if the $p$ value is larger than the Removal value. The Entry value must be less than the Removal value.

- **Use F value.** This option allows you to specify selection criteria based on the $F$ statistic associated with each field. The *F statistic* is a measure of how much each field contributes to the model. Fields will be added to the model only if the associated $F$ value is larger than the Entry value and will be removed only if the $F$ value is smaller than the Removal value. The Entry value must be greater than the Removal value.
Chapter 11

**Linear Regression Node Output Options**

Select the optional output you want to display in the advanced output of the generated linear regression model. To view the advanced output, browse the generated model and click the Advanced tab. For more information, see “Linear Regression Equation Advanced Output” in Chapter 12 on page 401.

**Figure 11-19**
Linear Regression advanced output options

```
<table>
<thead>
<tr>
<th>Model fit</th>
<th>Regression coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>R squared change</td>
<td>Confidence Interval</td>
</tr>
<tr>
<td>Selection criteria</td>
<td>Covariance matrix</td>
</tr>
<tr>
<td>Descriptives</td>
<td>Residuals</td>
</tr>
<tr>
<td>Part and partial correlations</td>
<td>Durbin-Watson</td>
</tr>
<tr>
<td>Collinearity Diagnostics</td>
<td></td>
</tr>
</tbody>
</table>
```

**Model fit.** Summary of model fit, including $R$-square. This represents the proportion of variance in the output field that can be explained by the input fields.

**R squared change.** The change in $R$-square at each step for Stepwise, Forwards, and Backwards estimation methods.

**Selection criteria.** Statistics estimating the information content of the model for each step of the model, to help evaluate model improvement. Statistics include the Akaike Information Criterion, Amemiya Prediction Criterion, Mallows' Prediction Criterion, and Schwarz Bayesian Criterion.

**Descriptives.** Basic descriptive statistics about the input and output fields.

**Part and partial correlations.** Statistics that help to determine importance and unique contributions of individual input fields to the model.

**Collinearity diagnostics.** Statistics that help to identify problems with redundant input fields.
Regression coefficients. Statistics for the regression coefficients:
- Confidence interval. The 95% confidence interval for each coefficient in the equation.
- Covariance matrix. The covariance matrix of the input fields.

Residuals. Statistics for the residuals, or the differences between predicted values and actual values.
- Durbin-Watson. The Durbin-Watson test of autocorrelation. This test detects effects of record order that can invalidate the regression model.

GRI Node

The Generalized Rule Induction (GRI) node discovers association rules in the data. Association rules are statements in the form

if antecedent(s) then consequent(s)

For example, if a customer purchases a razor and after-shave lotion, then that customer will purchase shaving cream with 80% confidence. GRI extracts a set of rules from the data, pulling out the rules with the highest information content. Information content is measured using an index that takes both the generality (support) and accuracy (confidence) of rules into account.

Requirements. To create a GRI ruleset, you need one or more In fields and one or more Out fields. Output fields (those with direction Out or Both) must be symbolic. Fields with direction None are ignored. Fields types must be fully instantiated before executing the node.

Strengths. Rulesets are usually fairly easy to interpret, in contrast to other methods such as neural networks. Rules in a ruleset can overlap such that some records may trigger more than one rule. This allows the ruleset to make rules more general than is possible with a decision tree. The GRI node can also handle multiple output fields. In contrast to Apriori, GRI can handle numeric as well as symbolic input fields.
**GRI Node Model Options**

Figure 11-20
GRI node options

![Diagram of the GRI node options interface](image)

**Model name.** Specify the name of the model to be produced.

- **Auto.** With this option selected, the model name will be generated automatically, based on the target or consequent field name(s). This is the default.

- **Custom.** Select this option to specify your own name for the generated model that will be created by this node.

**Minimum rule support.** You can also specify a support criterion (as a percentage). Support refers to the percentage of records in the training data for which the antecedents (the “if” part of the rule) are true. (Note that this definition of support differs from that used in the Sequence node. For more information, see “Sequence Node Model Options” on page 376.) If you are getting rules that apply to very small subsets of the data, try increasing this setting.

**Minimum rule confidence.** You can specify an accuracy criterion (as a percentage) for keeping rules in the ruleset. Rules with lower confidence than the specified criterion are discarded. If you are getting too many rules or uninteresting rules, try increasing this setting. If you are getting too few rules (or no rules at all), try decreasing this setting.
**Maximum number of antecedents.** You can specify the maximum number of antecedents for any rule. This is a way to limit the complexity of the rules. If the rules are too complex or too specific, try decreasing this setting. This setting also has a large influence on training time. If your ruleset is taking too long to train, try reducing this setting.

**Maximum number of rules.** This option determines the number of rules retained in the ruleset. Rules are retained in descending order of interest (as calculated by the GRI algorithm). Note that the ruleset may contain fewer rules than the number specified, especially if you use a stringent confidence or support criterion.

**Only true values for flags.** If this option is selected, only true values will appear in the resulting rules. This can help make rules easier to understand.

---

**Apriori Node**

The Apriori node also discovers association rules in the data. Apriori offers five different methods of selecting rules and uses a sophisticated indexing scheme to efficiently process large data sets.

**Requirements.** To create an Apriori ruleset, you need one or more In fields and one or more Out fields. Input and output fields (those with direction In, Out or Both) must be symbolic. Fields with direction None are ignored. Fields types must be fully instantiated before executing the node.

**Strengths.** For large problems, Apriori is generally faster to train than GRI. It also has no arbitrary limit on the number of rules that can be retained and can handle rules with up to 32 preconditions. Apriori offers five different training methods, allowing more flexibility in matching the data mining method to the problem at hand.
Apriori Node Model Options

Figure 11-21
Apriori node options

Model name. Specify the name of the model to be produced.
- **Auto.** With this option selected, the model name will be generated automatically, based on the target or consequent field name(s). This is the default.
- **Custom.** Select this option to specify your own name for the generated model that will be created by this node.

**Minimum rule support.** You can specify a support criterion for keeping rules in the ruleset. **Support** refers to the percentage of records in the training data for which the antecedents (the “if” part of the rule) are true. (Note that this definition of support differs from that used in the Sequence node. For more information, see “Sequence Node Model Options” on page 376.) If you are getting rules that apply to very small subsets of the data, try increasing this setting.

**Minimum rule confidence.** You can also specify a confidence criterion. **Confidence** is based on the records for which the rule's antecedents are true, and is the percentage of those records for which the consequent(s) are also true. In other words, it’s the percentage of predictions based on the rule that are correct. Rules with lower confidence than the specified criterion are discarded. If you are getting too many
rules, try increasing this setting. If you are getting too few rules (or no rules at all), try decreasing this setting.

**Maximum number of antecedents.** You can specify the maximum number of preconditions for any rule. This is a way to limit the complexity of the rules. If the rules are too complex or too specific, try decreasing this setting. This setting also has a large influence on training time. If your ruleset is taking too long to train, try reducing this setting.

**Only true values for flags.** If this option is selected, only true values will appear in the resulting rules. This can help make rules easier to understand.

**Optimize.** Select Speed to tune the algorithm to perform faster at the cost of more memory usage. Select Memory to sacrifice some speed in order to conserve memory.

**Apriori Node Expert Options**

For those with detailed knowledge of Apriori's operation, the following expert options allow you to fine-tune the induction process. To access expert options, set the Mode to Expert on the Expert tab.

**Figure 11-22**

*Apriori expert options*
**Evaluation measure.** Apriori supports five methods of evaluating potential rules.

- **Rule Confidence.** The default method uses the confidence (or accuracy) of the rule to evaluate rules. For this measure, the Evaluation measure lower bound is disabled, since it is redundant with the Minimum rule confidence option on the Model tab. For more information, see “Apriori Node Model Options” on page 342.

- **Confidence Difference.** (Also called **absolute confidence difference to prior.**) This evaluation measure is the absolute difference between the rule's confidence and its prior confidence. This option prevents bias where the outcomes are not evenly distributed. This helps prevent “obvious” rules from being kept. For example, it may be the case that 80% of customers buy your most popular product. A rule that predicts buying that popular product with 85% accuracy doesn't add much to your knowledge, even though 85% accuracy may seem quite good on an absolute scale. Set the evaluation measure lower bound to the minimum difference in confidence for which you want rules to be kept.

- **Confidence Ratio.** (Also called **difference of confidence quotient to 1.**) This evaluation measure is the ratio of rule confidence to prior confidence (or, if the ratio is greater than one, its reciprocal) subtracted from 1. Like Confidence Difference, this method takes uneven distributions into account. It is especially good at finding rules that predict rare events. For example, suppose that there is a rare medical condition that occurs in only 1% of patients. A rule that is able to predict this condition 10% of the time is a great improvement over random guessing, even though on an absolute scale, 10% accuracy might not seem very impressive. Set the evaluation measure lower bound to the difference for which you want rules to be kept.

- **Information Difference.** (Also called **information difference to prior.**) This measure is based on the information gain measure. If the probability of a particular consequent is considered as a logical value (a **bit**), then the information gain is the proportion of that bit that can be determined, based on the antecedents. The information difference is the difference between the information gain, given the antecedents, and the information gain, given only the prior confidence of the consequent. An important feature of this method is that it takes support into account so that rules that cover more records are preferred for a given level of confidence. Set the evaluation measure lower bound to the information difference for which you want rules to be kept.
Note: Because the scale for this measure is somewhat less intuitive than the other scales, you may need to experiment with different lower bounds to get a satisfactory ruleset.

- **Normalized Chi-square.** (Also called normalized chi-squared measure.) This measure is a statistical index of association between antecedents and consequents. The measure is normalized to take values between 0 and 1. This measure is even more strongly dependent on support than the information difference measure. Set the evaluation measure lower bound to the information difference for which you want rules to be kept.

Note: As with the information difference measure, the scale for this measure is somewhat less intuitive than the other scales, so you may need to experiment with different lower bounds to get a satisfactory ruleset.

**K-Means Node**

The K-Means node provides a method of cluster analysis. It can be used to cluster the data set into distinct groups when you don't know what those groups are at the beginning. Unlike most learning methods in Clementine, K-Means models do not use a target field. This type of learning, with no target field, is called unsupervised learning. Instead of trying to predict an outcome, K-Means tries to uncover patterns in the set of input fields. Records are grouped so that records within a group or cluster tend to be similar to each other, but records in different groups are dissimilar.

K-Means works by defining a set of starting cluster centers derived from data. It then assigns each record to the cluster to which it is most similar, based on the record's input field values. After all cases have been assigned, the cluster centers are updated to reflect the new set of records assigned to each cluster. The records are then checked again to see whether they should be reassigned to a different cluster, and the record assignment/cluster iteration process continues until either the maximum number of iterations is reached, or the change between one iteration and the next fails to exceed a specified threshold.

Note: The resulting model depends to a certain extent on the order of the training data. Reordering the data and rebuilding the model may lead to a different final cluster model.

**Requirements.** To train a K-Means model, you need one or more In fields. Fields with direction Out, Both, or None are ignored.
**Strengths.** You do not need to have data on group membership to build a K-Means model. The K-Means model is often the fastest method of clustering for large data sets.

**K-Means Node Model Options**

Figure 11-23  
*K-Means node options*

- **Model name.** Specify the name of the model to be produced.
  - **Auto.** With this option selected, the model name will be “Kmeans.” This is the default.
  - **Custom.** Select this option to specify your own name for the generated model that will be created by this node.

- **Specified number of clusters.** Specify the number of clusters to generate. The default is 5.

- **Generate distance field.** If this option is selected, the generated model will include a field containing the distance of each record from the center of its assigned cluster.

- **Show cluster proximity.** Select this option to include information about distances between cluster centers in the generated model output.
**Cluster display.** Specify the format for the generated cluster membership field. Cluster membership can be indicated as a String with the specified Label prefix (for example "Cluster 1", "Cluster 2", etc.), or as a Number.

*Note:* Use binary set encoding, an option available in previous versions of Clementine, has been removed. In some situations, that option tended to distort distance information between records and was thus unsuitable for use with K-Means models, which rely heavily on such distance information. If you want to include set fields in your model but are having memory problems in building the model or the model is taking too long to build, consider recoding large set fields to reduce the number of values or using a different field with fewer values as a proxy for the large set. For example, if you are having a problem with a *product_id* field containing values for individual products, you might consider removing it from the model and adding a less detailed *product_category* field instead.

**K-Means Node Expert Options**

For those with detailed knowledge of $k$-means clustering, expert options allow you to fine-tune the training process. To access expert options, set the Mode to Expert on the Expert tab.

*Figure 11-24*

*K-Means expert options*
**Stop on.** Specify the stopping criterion to be used in training the model. The Default stopping criterion is 20 iterations or change < .000001, whichever occurs first. Select Custom to specify your own stopping criteria.

- **Maximum Iterations.** This option allows you to stop model training after the number of iterations specified.
- **Change tolerance.** This option allows you to stop model training when the largest change in cluster centers for an iteration is less than the level specified.

**Encoding value for sets.** Specify a value between 0 and 1.0 to use for recoding set fields as groups of numeric fields. The default value is the square root of 0.5 (approximately 0.707107), to provide the proper weighting for recoded flag fields. Values closer to 1.0 will weight set fields more heavily than numeric fields.

### Logistic Regression Node

Logistic regression, also known as nominal regression, is a statistical technique for classifying records based on values of input fields. It is analogous to linear regression but takes a symbolic target field instead of a numeric one.

Logistic regression works by building a set of equations that relate the input field values to the probabilities associated with each of the output field categories. Once the model is generated, it can be used to estimate probabilities for new data. For each record, a probability of membership is computed for each possible output category. The target category with the highest probability is assigned as the predicted output value for that record.

**Requirements.** To build a logistic regression model, you need one or more In fields and exactly one symbolic Out field. Fields set to Both or None are ignored. Fields used in the model must have their types fully instantiated.

**Strengths.** Logistic regression models are often quite accurate. They can handle symbolic and numeric input fields. They can give predicted probabilities for all target categories so that a “second-best guess” can easily be identified. They can also perform automatic field selection for the logistic model.

When processing large data sets, you can improve performance noticeably by disabling the Likelihood ratio test, an advanced output option. For more information, see “Logistic Regression Node Output Options” on page 356.
Logistic Regression Node Model Options

Figure 11-25
Logistic Regression node options

Model name. Specify the name of the model to be produced.

- **Auto.** With this option selected, the model name will be generated automatically, based on the target field name. This is the default.

- **Custom.** Select this option to specify your own name for the generated model that will be created by this node.

Method. Specify the method to be used in building the logistic regression model.

- **Enter.** This is the default method, which enters all of the terms into the equation directly. No field selection is performed in building the model.

- **Stepwise.** The Stepwise method of field selection builds the equation in steps, as the name implies. The initial model is the simplest model possible, with no model terms (except the constant) in the equation. At each step, terms that have not yet been added to the model are evaluated, and if the best of those terms adds
significantly to the predictive power of the model, it is added. In addition, terms that are currently in the model are reevaluated to determine if any of them can be removed without significantly detracting from the model. If so, they are removed. Then the process repeats, and other terms are added and/or removed. When no more terms can be added to improve the model and no more terms can be removed without detracting from the model, the final model is generated.

**Forwards.** The Forwards method of field selection is similar to the Stepwise method in that the model is built in steps. However, with this method, the initial model is the simplest model, and only the constant and terms can be added to the model. At each step, terms not yet in the model are tested based on how much they would improve the model, and the best of those is added to the model. When no more terms can be added or the best candidate term does not produce a large enough improvement in the model, the final model is generated.

**Backwards.** The Backwards method is essentially the opposite of the Forwards method. With this method, the initial model contains all of the terms as predictors, and terms can only be removed from the model. Model terms that contribute little to the model are removed one by one until no more terms can be removed without significantly worsening the model, yielding the final model.

**Backwards Stepwise.** The Backwards Stepwise method is essentially the opposite of the Stepwise method. With this method, the initial model contains all of the terms as predictors. At each step, terms in the model are evaluated, and any terms that can be removed without significantly detracting from the model are removed. In addition, previously removed terms are reevaluated to determine if the best of those terms adds significantly to the predictive power of the model. If so, it is added back into the model. When no more terms can be removed without significantly detracting from the model and no more terms can be added to improve the model, the final model is generated.

*Note:* The automatic methods, including Stepwise, Forwards, and Backwards, are highly adaptable learning methods and have a strong tendency to overfit the training data. When using these methods, it is especially important to verify the validity of the resulting model with a hold-out test sample or new data.

**Model type.** There are three options for defining the terms in the model. **Main Effects** models include only the input fields individually and do not test interactions (multiplicative effects) between input fields. **Full Factorial** models include all interactions as well as the input field main effects. Full factorial models are better able to capture complex relationships but are also much more difficult to interpret and
are more likely to suffer from overfitting. Because of the potentially large number of possible combinations, automatic field selection methods (methods other than Enter) are disabled for full factorial models. Custom models include only the terms (main effects and interactions) that you specify. When selecting this option, use the Model Terms list to add or remove terms in the model.

Model Terms. When building a Custom model, you will need to explicitly specify the terms in the model. The list shows the current set of terms for the model. The buttons on the right side of the Model Terms list allow you to add and remove model terms.

- To add terms to the model, click the Add new model terms button.
- To delete terms, select the desired terms and click the Delete selected model terms button.

Include constant in equation. This option determines whether the resulting equations will include a constant term. In most situations, you should leave this option selected.

Adding Terms to a Logistic Regression Model

When requesting a custom logistic regression model, you can add terms to the model by clicking the Add new model terms button on the Logistic Regression Model tab. A new dialog box opens in which you can specify terms.
Chapter 11

**Figure 11-26**
*Logistic Regression New Terms dialog box*

![New Terms dialog box](image)

**Type of term to add.** There are several ways to add terms to the model, based on the selection of input fields in the Available fields list.

- **Single interaction.** Inserts the term representing the interaction of all selected fields.

- **Main effects.** Inserts one main effect term (the field itself) for each selected input field.

- **All 2-way interactions.** Inserts a two-way interaction term (the product of the input fields) for each possible pair of selected input fields. For example, if you have selected input fields A, B, and C in the Available fields list, this method will insert the terms $A \times B$, $A \times C$, and $B \times C$.

- **All 3-way interactions.** Inserts a three-way interaction term (the product of the input fields) for each possible combination of selected input fields, taken three at a time. For example, if you have selected input fields $A$, $B$, $C$, and $D$ in the
Available fields list, this method will insert the terms $A \times B \times C$, $A \times B \times D$, $A \times C \times D$, and $B \times C \times D$.

**All 4-way interactions.** Inserts a four-way interaction term (the product of the input fields) for each possible combination of selected input fields, taken four at a time. For example, if you have selected input fields $A$, $B$, $C$, $D$, and $E$ in the Available fields list, this method will insert the terms $A \times B \times C \times D$, $A \times B \times C \times E$, $A \times B \times D \times E$, $A \times C \times D \times E$, and $B \times C \times D \times E$.

**Available fields.** Lists the available input fields to be used in constructing model terms.

**Preview.** Shows the terms that will be added to the model if you click Insert, based on the selected fields and the term type selected above.

**Insert.** Inserts terms in the model based on the current selection of fields and term type and closes the dialog box.

---

**Logistic Regression Node Expert Options**

For those with detailed knowledge of logistic regression, expert options allow you to fine-tune the training process. To access expert options, set the Mode to Expert on the Expert tab.
**Logistic Regression expert options**

**Figure 11-27**

*You can specify a dispersion scaling value that will be used to correct the estimate of the parameter covariance matrix. Pearson estimates the scaling value using the Pearson chi-square statistic. Deviance estimates the scaling value using the deviance function (likelihood-ratio chi-square) statistic. You can also specify your own User-defined scaling value. It must be a positive numeric value.***

**Append all probabilities.** If this option is selected, probabilities for each category of the output field will be added to each record processed by the node. If not selected, the probability of only the predicted category is added.

**Singularity tolerance.** Specify the tolerance used in checking for singularities.

**Convergence.** These options allow you to control the parameters for model convergence. For more information, see “Logistic Regression Node Convergence Options” on page 355.
**Output.** These options allow you to request additional statistics that will appear in the advanced output of the generated model built by the node. For more information, see “Logistic Regression Node Output Options” on page 356.

**Stepping.** These options allow you to control the criteria for adding and removing fields with the Stepwise, Forwards, Backwards, or Backwards Stepwise estimation methods. (The button is disabled if the Enter method is selected.) For more information, see “Logistic Regression Node Stepping Options” on page 357.

---

**Logistic Regression Node Convergence Options**

You can set the convergence parameters for logistic regression model estimation.

Figure 11-28
*Logistic Regression convergence options*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum iterations</td>
<td>100</td>
</tr>
<tr>
<td>Maximum step-halving</td>
<td>5</td>
</tr>
<tr>
<td>Log-likelihood convergence</td>
<td>1.0E-2</td>
</tr>
<tr>
<td>Parameter convergence</td>
<td>1.0E-6</td>
</tr>
<tr>
<td>Delta</td>
<td>0</td>
</tr>
</tbody>
</table>

**Maximum iterations.** Specify the maximum number of iterations for estimating the model.

**Maximum step-halving.** Step-halving is a technique used by logistic regression to deal with complexities in the estimation process. Under normal circumstances, you should use the default setting.

**Log-likelihood convergence.** Iterations stop if the relative change in the log-likelihood is less than this value. The criterion is not used if the value is 0.

**Parameter convergence.** Iterations stop if the absolute change or relative change in the parameter estimates is less than this value. The criterion is not used if the value is 0.
**Delta.** You can specify a value between 0 and 1 to be added to each empty cell (combination of input field and output field values). This can help the estimation algorithm deal with data where there are many possible combinations of field values relative to the number of records in the data. The default is 0.

**Logistic Regression Node Output Options**

Figure 11-29

*Logistic Regression output options*

Select the optional output you want to display in the advanced output of the generated logistic regression model. To view the advanced output, browse the generated model and select the **Advanced** tab. For more information, see “Logistic Regression Advanced Output” in Chapter 12 on page 397.

**Summary statistics.** The Cox and Snell, Nagelkerke, and McFadden \( R \)-square measures of model fit. These statistics are in some ways analogous to the \( R \)-square statistic in linear regression.

**Likelihood ratio test.** Tests of whether the coefficients of the model effects are statistically different from 0. Significant input fields are those with very small significance levels in the output (labeled **Sig**.).

*Note:* This option greatly increases the processing time required to build a logistic regression model. If your model is taking too long to build, consider disabling this option.

**Asymptotic correlation.** The estimated correlation matrix of the coefficient estimates.

**Goodness-of-fit chi-square statistics.** Pearson's and likelihood-ratio chi-square statistics. These statistics test the overall fit of the model to the training data.
**Iteration history.** Specify the interval for printing iteration status in the advanced output.

**Parameter estimates.** Estimates of the equation coefficients.

- **Confidence interval.** The confidence intervals for coefficients in the equations. Specify the level of the confidence interval (the default is 95%).

**Asymptotic covariance.** The estimated covariance matrix of the parameter estimates.

**Classification table.** Table of the observed versus predicted responses.

---

**Logistic Regression Node Stepping Options**

*Figure 11-30  
*Linear Regression stepping options*

**Number of terms in model.** You can specify the minimum number of terms in the model for Backwards and Backwards Stepwise models and the maximum number of terms for Forwards and Stepwise models. If you specify a minimum value greater than 0, the model will include that many terms, even if some of the terms would have been removed based on statistical criteria. The minimum setting is ignored for Forwards, Stepwise, and Enter models. If you specify a maximum, some terms may be omitted from the model, even though they would have been selected based
on statistical criteria. The Maximum setting is ignored for Backwards, Backwards Stepwise, and Enter models.

**Significance thresholds for LR criteria.** This option allows you to specify selection criteria based on the statistical probability (the $p$ value) associated with each field. Fields will be added to the model only if the associated $p$ value is smaller than the Entry value and will be removed only if the $p$ value is larger than the Removal value. The Entry value must be smaller than the Removal value.

**Requirements for entry or removal.** For some applications, it doesn't make mathematical sense to add interaction terms to the model unless the model also contains the lower-order terms for the fields involved in the interaction term. For example, it may not make sense to include $A * B$ in the model unless $A$ and $B$ also appear in the model. These options let you determine how such dependencies are handled during stepwise term selection.

**Hierarchy for discrete effects.** Higher-order effects (interactions involving more fields) will enter the model only if all lower-order effects (main effects or interactions involving fewer fields) for the relevant fields are already in the model, and lower-order effects will not be removed if higher-order effects involving the same fields are in the model. Applies only to discrete fields. For more information, see “Data Types” in Chapter 7 on page 150.

- **Hierarchy for all effects.** As described above, except applies to all input fields.

- **Containment for all effects.** Effects can appear in the model only if all of the effects contained in the effect also appear in the model. This option is similar to the Hierarchy for all effects option except that range fields are treated somewhat differently. For an effect to contain another effect, the contained (lower-order) effect must include all of the range fields involved in the containing (higher-order) effect, and the contained effect’s discrete fields must be a subset of those in the containing effect. For example, if $A$ and $B$ are discrete fields and $X$ is a range field, then the term $A * B * X$ contains the terms $A * X$ and $B * X$.

- **None.** No relationships are enforced; terms are added to and removed from the model independently.
Factor Analysis/PCA Node

The Factor/PCA node provides powerful data reduction techniques to reduce the complexity of your data. Two similar but distinct approaches are provided.

- **Principal components analysis (PCA)** finds linear combinations of the input fields that do the best job of capturing the variance in the entire set of fields, where the components are orthogonal (perpendicular) to each other. PCA focuses on all variance, including both shared and unique variance.

- **Factor analysis** attempts to identify underlying concepts, or factors, that explain the pattern of correlations within a set of observed fields. Factor analysis focuses on shared variance only. Variance that is unique to specific fields is not considered in estimating the model. Several methods of factor analysis are provided by the Factor/PCA node.

For both approaches, the goal is to find a small number of derived fields that effectively summarize the information in the original set of fields.

**Requirements.** Only numeric fields can be used in a Factor/PCA model. To estimate a factor analysis or PCA, you need one or more In fields. Fields with direction Out, Both, or None are ignored, as are non-numeric fields.

**Strengths.** Factor analysis and PCA can effectively reduce the complexity of your data without sacrificing much of the information content. These techniques can help you build more robust models that execute more quickly than would be possible with the raw input fields.
**Factor Analysis/PCA Node Model Options**

**Figure 11-31**
*Factor/PCA node options*

- **Model name.** Specify the name of the model to be produced.
  - **Auto.** With this option selected, the model name will be “Factor.” This is the default.
  - **Custom.** Select this option to specify your own name for the generated model that will be created by this node.

- **Extraction Method.** Specify the method to be used for data reduction.
  - **Principal Components.** This is the default method, which uses PCA to find components that summarize the input fields.
  - **Unweighted Least Squares.** This factor analysis method works by finding the set of factors that is best able to reproduce the pattern of relationships (correlations) among the input fields.
- **Generalized Least Squares.** This factor analysis method is similar to unweighted least squares, except that it uses weighting to de-emphasize fields with a lot of unique (unshared) variance.

- **Maximum Likelihood.** This factor analysis method produces factor equations that are most likely to have produced the observed pattern of relationships (correlations) in the input fields, based on assumptions about the form of those relationships. Specifically, the method assumes that the training data follow a multivariate normal distribution.

- **Principal Axis Factoring.** This factor analysis method is very similar to the principal components method, except that it focuses on shared variance only.

- **Alpha Factoring.** This factor analysis method considers the fields in the analysis to be a sample from the universe of potential input fields. It maximizes the statistical reliability of the factors.

- **Image Factoring.** This factor analysis method uses data estimation to isolate the common variance and find factors that describe it.

**Factor Analysis/PCA Node Expert Options**

For those with detailed knowledge of factor analysis and PCA, expert options allow you to fine-tune the training process. To access expert options, set the Mode to Expert on the Expert tab.
**Missing values.** By default, Clementine will use only records that have valid values for all fields used in the model. (This is sometimes called *listwise deletion* of missing values). If you have a lot of missing data, you may find that this approach eliminates too many records, leaving you without enough data to generate a good model. In such cases, you can deselect the *Only use complete records* option. Clementine will then attempt use as much information as possible to estimate the model, including records where some of the fields have missing values. (This is sometimes called *pairwise deletion* of missing values.) However, in some situations, using incomplete records in this manner can lead to computational problems in estimating the model.

**Fields.** Specify whether to use the *Correlation matrix* (the default) or the *Covariance matrix* of the input fields in estimating the model.

**Maximum iterations for convergence.** Specify the maximum number of iterations for estimating the model.
**Extract factors.** There are two ways to select the number of factors to extract from the input fields.

- **Eigenvalues over.** This option will retain all factors or components with eigenvalues larger than the specified criterion. **Eigenvalues** measure the ability of each factor or component to summarize variance in the set of input fields. The model will retain all factors or components with eigenvalues greater than the specified value when using the correlation matrix. When using the covariance matrix, the criterion is the specified value times the mean eigenvalue. That scaling gives this option a similar meaning for both types of matrix.

- **Maximum number.** This option will retain the specified number of factors or components in descending order of eigenvalues. In other words, the factors or components corresponding to the $n$ highest eigenvalues are retained, where $n$ is the specified criterion. The default extraction criterion is five factors/components.

**Component/factor matrix format.** These options control the format of the factor matrix (or component matrix for PCA models).

- **Sort values.** If this option is selected, factor loadings in the model output will be sorted numerically.

- **Hide values below.** If this option is selected, scores below the specified threshold will be hidden in the matrix to make it easier to see the pattern in the matrix.

**Rotation.** These options allow you to control the rotation method for the model. For more information, see “Factor/PCA Node Rotation Options” on page 363.

**Factor/PCA Node Rotation Options**

![Factor/PCA rotation options](image)
In many cases, mathematically *rotating* the set of retained factors can increase their usefulness and especially their interpretability. Select a rotation method:

- **None.** The default option. No rotation is used.
- **Varimax.** An orthogonal rotation method that minimizes the number of fields with high loadings on each factor. It simplifies the interpretation of the factors.
- **Direct oblimin.** A method for oblique (nonorthogonal) rotation. When Delta equals 0 (the default), solutions are oblique. As delta becomes more negative, the factors become less oblique. To override the default delta of 0, enter a number less than or equal to 0.8.
- **Quartimax.** An orthogonal method that minimizes the number of factors needed to explain each field. It simplifies the interpretation of the observed fields.
- **Equamix.** A rotation method that is a combination of the Varimax method, which simplifies the factors, and the Quartimax method, which simplifies the fields. The number of fields that load highly on a factor and the number of factors needed to explain a field are minimized.
- **Promax.** An oblique rotation, which allows factors to be correlated. It can be calculated more quickly than a Direct Oblimin rotation, so it can be useful for large data sets. Kappa controls the obliqueness (the extent to which factors can be correlated) of the solution.

**TwoStep Cluster Node**

The TwoStep Cluster node provides a form of *cluster analysis*. It can be used to cluster the data set into distinct groups when you don't know what those groups are at the beginning. As with Kohonen nodes and K-Means nodes, TwoStep Cluster models do *not* use a target field. Instead of trying to predict an outcome, TwoStep Cluster tries to uncover patterns in the set of input fields. Records are grouped so that records within a group or cluster tend to be similar to each other, but records in different groups are dissimilar.

TwoStep Cluster is a two-step clustering method. The first step makes a single pass through the data, during which it compresses the raw input data into a manageable set of subclusters. The second step uses a hierarchical clustering method to progressively merge the subclusters into larger and larger clusters, without requiring another pass through the data. Hierarchical clustering has the advantage of not requiring the number of clusters to be selected ahead of time. Many hierarchical clustering
methods start with individual records as starting clusters, and merge them recursively to produce ever larger clusters. Though such approaches often break down with large amounts of data, TwoStep's initial preclustering makes hierarchical clustering fast even for large data sets.

*Note:* The resulting model depends to a certain extent on the order of the training data. Reordering the data and rebuilding the model may lead to a different final cluster model.

**Requirements.** To train a TwoStep Cluster model, you need one or more *In* fields. Fields with direction *Out, Both, or None* are ignored. The TwoStep Cluster algorithm does not handle missing values. Records with blanks for any of the input fields will be ignored when building the model.

**Strengths.** TwoStep Cluster can handle mixed field types and is able to handle large data sets efficiently. It also has the ability to test several cluster solutions and choose the best, so you don't need to know how many clusters to ask for at the outset. TwoStep Cluster can be set to automatically exclude *outliers*, or extremely unusual cases that can contaminate your results.
TwoStep Cluster Node Model Options

**Figure 11-34**
TwoStep Cluster node options

![TwoStep Cluster node options](image)

- **Model name.** Specify the name of the model to be produced.
  - **Auto.** With this option selected, the model name will be “TwoStep.” This is the default.
  - **Custom.** Select this option to specify your own name for the generated model that will be created by this node.

- **Standardize numeric fields.** By default, TwoStep will standardize all numeric input fields to the same scale, with a mean of 0 and a variance of 1. To retain the original scaling for numeric fields, deselect this option. Symbolic fields are not affected.

- **Exclude outliers.** If you select this option, records that don't appear to fit into a substantive cluster will be automatically excluded from the analysis. This prevents such cases from distorting the results.

Outlier detection occurs during the pre-clustering step. When this option is selected, subclusters with few records relative to other subclusters are considered potential outliers, and the tree of subclusters is rebuilt excluding those records. Some of those
Modeling Nodes

potential outlier records may be added to the rebuilt subclusters, if they are similar enough to any of the new subcluster profiles. The rest of the potential outliers that cannot be merged are considered outliers, and are added to a “noise” cluster and excluded from the hierarchical clustering step.

When scoring data with a TwoStep model that uses outlier handling, new cases that are more than a certain threshold distance (based on the log-likelihood) from the nearest substantive cluster are considered outliers and are assigned to the “noise” cluster.

**Cluster label.** Specify the format for the generated cluster membership field. Cluster membership can be indicated as a String with the specified Label prefix (for example "Cluster 1", "Cluster 2", etc.), or as a Number.

**Automatically calculate number of clusters.** TwoStep cluster can very rapidly analyze a large number of cluster solutions to choose the optimal number of clusters for the training data. Specify a range of solutions to try by setting the Maximum and the Minimum number of clusters. TwoStep uses a two-stage process to determine the optimal number of clusters. In the first stage, an upper bound on the number of clusters in the model is selected based on the change in the Bayes Information Criterion (BIC) as more clusters are added. In the second stage, the change in the minimum distance between clusters is found for all models with fewer clusters than the minimum-BIC solution. The largest change in distance is used to identify the final cluster model.

**Specify number of clusters.** If you know how many clusters to include in your model, select this option and enter the number of clusters.

**C&R Tree Node**

The Classification and Regression (C&R) Tree node is a tree-based classification and prediction method. Similar to C5.0, this method uses recursive partitioning to split the training records into segments with similar output field values. C&R Tree starts by examining the input fields to find the best split, measured by the reduction in an impurity index that results from the split. The split defines two subgroups, each of which is subsequently split into two more subgroups, and so on, until one of the stopping criteria is triggered.
Chapter 11

Note: The C&R Tree node replaces the Build Rule node from earlier versions of Clementine. If you load a file containing a Build Rule node, it will be replaced with a C&R Tree node, as the Build Rule node is no longer supported. Generated models created by Build Rule, however, will continue to operate in the usual way and will not be replaced.

Requirements. To train a C&R Tree model, you need one or more In fields and exactly one Out field. The output field can be numeric or symbolic. Fields set to Both or None are ignored. Fields used in the model must have their types fully instantiated.

Strengths. C&R Tree models are quite robust in the presence of problems such as missing data and large numbers of fields. They usually do not require long training times to estimate. In addition, C&R Tree models tend to be easier to understand than some other model types—the rules derived from the model have a very straightforward interpretation. Unlike C5.0, C&R Tree can accommodate numeric as well as symbolic output fields.

C&R Tree Node Model Options

Figure 11-35
C&R Tree node options
**Model name.** Specify the name of the model to be produced.

- **Auto.** With this option selected, the model name will be generated automatically, based on the target field name. This is the default.
- **Custom.** Select this option to specify your own name for the generated model that will be created by this node.

**Maximum tree depth.** Specify the maximum number of levels below the root node (the number of times the sample will be split recursively).

---

**C&R Tree Node Expert Options**

For those with detailed knowledge of C&R Tree models, expert options allow you to fine-tune the model-building process. To access expert options, set the Mode to Expert on the Expert tab.

**Figure 11-36**
*C&R Tree expert options*

**Maximum surrogates.** Surrogates are a method for dealing with missing values. For each split in the tree, C&R Tree identifies the input fields that are most similar to the selected split field. Those fields are the surrogates for that split. When a record
must be classified but has a missing value for a split field, its value on a surrogate field can be used to make the split. Increasing this setting will allow more flexibility to handle missing values, but may also lead to increased memory usage and longer training times.

**Minimum change in impurity.** Specify the minimum change in impurity to create a new split in the tree. If the best split for a branch reduces the impurity of the tree by less than the specified amount, the split will not be made.

**Impurity measure for categorical targets.** These options allow you to select the method used to measure the impurity of the tree. **Impurity** refers to the extent to which subgroups defined by the tree have a wide range of output field values within each group. The goal of the tree is to create subgroups such that each subgroup tends to have the same or similar output values—in other words, to minimize the impurity of the tree. **Gini** is a general impurity measure based on probabilities of category membership for the branch. **Twoing** is an impurity measure that emphasizes the binary split and is more likely to lead to approximately equal-sized branches from a split. This option only affects symbolic target fields; numeric target fields always use the least squared deviation impurity measure.

**Stopping.** These options allow you to control the criteria for deciding when to stop splitting nodes in the tree. For more information, see “C&R Tree Node Stopping Options” on page 371.

**Prune tree.** Pruning consists of removing bottom-level splits that do not contribute significantly to the accuracy of the tree. Pruning can help simplify the tree, making it easier to interpret and, in some cases, improving generalization. If you want the full tree without pruning, deselect this option.

- **Use standard error rule.** Allows you to specify a more liberal pruning rule. The standard error rule allows C&R Tree to select the simplest tree whose risk estimate is close to (but possibly greater than) that of the subtree with the smallest risk. The **multiplier** indicates the size of the allowable difference in the risk estimate between the pruned tree and the tree with the smallest risk in terms of the risk estimate. For example, if you specify “2,” a tree whose risk estimate is \(2 \times \text{standard error}\) larger than that of the full tree could be selected.

**Priors.** These options allow you to set prior probabilities for target categories. For more information, see “C&R Tree Node Prior Probability Options” on page 372.
**C&R Tree Node Stopping Options**

*Figure 11-37  
C&R Tree stopping options*

These options control how the tree is constructed. Stopping rules determine when to stop splitting specific branches of the tree. Set the minimum branch sizes to prevent splits that would create very small subgroups. Minimum records in parent branch will prevent a split if the number of records in the node to be split (the **parent** node) is less than the specified value. Minimum records in child branch will prevent a split if the number of records in any branch created by the split (the **child** node) would be less than the specified value.

- **Use percentage.** Allows you to specify sizes in terms of percentage of overall training data.
- **Use absolute value.** Allows you to specify sizes as the absolute numbers of records.
These options allow you to specify prior probabilities for categories when predicting a symbolic target field. **Prior probabilities** are estimates of the overall relative frequency for each target category in the population from which the training data are drawn. In other words, they are the probability estimates you would make for each possible target value *prior* to knowing anything about predictor values. There are three methods of setting priors.

**Based on training data.** This is the default. Prior probabilities are based on the relative frequencies of the categories in the training data.

**Equal for all classes.** Prior probabilities for all categories are defined as $1/k$, where $k$ is the number of target categories.

**Custom.** You can specify your own prior probabilities. Starting values for prior probabilities are set as equal for all classes. You can adjust the probabilities for individual categories to user-defined values. To adjust a specific category's probability, select the probability cell in the table corresponding to the desired category, delete the contents of the cell, and enter the desired value.
The prior probabilities for all categories should sum to 1.0 (the \textbf{probability constraint}). If they do not sum to 1.0, Clementine will give a warning and offer to automatically normalize the values. This automatic adjustment preserves the proportions across categories while enforcing the probability constraint. You can perform this adjustment at any time by clicking the \textit{Normalize} button. To reset the table to equal values for all categories, click the \textit{Equalize} button.

\textbf{Adjust priors using misclassification costs.} This option allows you to adjust the priors, based on misclassification costs. This enables you to incorporate cost information into the tree-growing process directly for trees that use the Twoing impurity measure. (When this option is not selected, cost information is used only in classifying records and calculating risk estimates for trees, based on the Twoing measure.)

\section*{Sequence Node}

The sequence node discovers patterns in sequential or time-oriented data. The Sequence node extracts a set of predictable sequences from sequential data. The elements of a sequence are \textbf{item sets}, or sets of one or more items that constitute a single transaction. For example, if a person goes to the store and purchases bread and milk and then a few days later returns to the store and purchases some cheese, that person's buying activity can be represented as two item sets. The first item set contains bread and milk, and the second one contains cheese. A \textbf{sequence} is a list of item sets that tend to occur in a predictable order. The sequence node detects frequent sequences and creates a generated model node that can be used to make predictions.

\textbf{Requirements.} To create a Sequence ruleset, you need to specify an ID field, an optional Time field, and one or more Content fields. Note that these settings must be made on the Fields tab of the Modeling node; they cannot be read from an upstream Type node. The ID field can have any direction or type. If you specify a time field, it can have any direction but must be numeric, date, time, or timestamp. If you do not specify a time field, the Sequence node will use an implied time stamp, in effect using row numbers as time values. Content fields can have any type and direction, but all content fields must be of the same type. If they are numeric, they must be integer ranges (not real ranges).

\textbf{Strengths.} The Sequence node is based on the CARMA association rule algorithm, which uses an efficient two-pass method for finding sequences. In addition, the generated model node created by a Sequence node can be inserted into a data stream...
to create predictions. The generated model node can also generate SuperNodes for detecting and counting specific sequences and for making predictions based on specific sequences.

**Sequence Node Field Options**

Figure 11-39
Sequence node field options

Before executing a Sequence node, you must specify ID and content fields on the Fields tab of the Sequence node. If you want to use a time field, you also need to specify that here.

**Content fields.** Specify the content field(s) for the model. These fields contain the events of interest in sequence modeling.

**ID field.** Select an ID field from the list. Numeric or symbolic fields can be used as the ID field. Each unique value of this field should indicate a specific unit of analysis. For example, in a market basket application, each ID might represent a single
customer. For a Web log analysis application, each ID might represent a computer (by IP address) or a user (by login data).

- **IDs are contiguous.** If your data are presorted so that all records with the same ID appear together in the data stream, select this option to speed up processing. If your data are not presorted (or you are not sure), leave this option unselected, and the Sequence node will sort the data automatically.

*Note:* If your data are not sorted and you select this option, you may get invalid results in your Sequence model.

**Time field.** If you want to use a field in the data to indicate event times, select Use time field and specify the field to be used. The time field must be numeric, date, time, or timestamp. If no time field is specified, records are assumed to arrive from the data source in sequential order, and record numbers are used as time values (the first record occurs at time "1"; the second, at time "2"; etc.).

**Content fields.** Specify the content field(s) for the model. These fields contain the events of interest in sequence modeling.

The Sequence node can handle data in either of two formats:

- **Tabular** data has items represented by separate flag fields, where each flag field represents the presence or absence of a specific item.

- **Transactional** data has one or more content fields for items.

The content field(s) contain values indicating which items belong to the transaction. These can be numeric or symbolic. If you use multiple fields with transactional data, the items specified in these fields for a particular record are assumed to represent items found in a single transaction with a single time stamp.

Following is an example of **Tabular** data:

<table>
<thead>
<tr>
<th>Customer</th>
<th>jam</th>
<th>bread</th>
<th>milk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Chapter 11

Here are the same data shown in **Transactional** format:

<table>
<thead>
<tr>
<th>Customer</th>
<th>Time</th>
<th>Purchase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>jam</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>milk</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>jam</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>bread</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>jam</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>bread</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>milk</td>
</tr>
</tbody>
</table>

**Sequence Node Model Options**

**Figure 11-40**

*Sequence node options*

**Model name.** Specify the name of the model to be produced.

- **Auto.** With this option selected, the model name will be “Sequence.” This is the default.

- **Custom.** Select this option to specify your own name for the generated model that will be created by this node.
**Minimum rule support (%)**. You can specify a support criterion. **Support** refers to the proportion of IDs in the training data that contain the entire sequence. (Note that this definition of support differs from that used in the GRI and Apriori nodes.) If you want to focus on more common sequences, increase this setting.

**Minimum rule confidence (%)**. You can specify a confidence criterion for keeping sequences in the sequence set. **Confidence** refers to the percentage of the IDs where a correct prediction is made, out of all the IDs for which the rule makes a prediction. It is calculated as the number of IDs for which the entire sequence is found divided by the number of IDs for which the antecedents are found, based on the training data. Sequences with lower confidence than the specified criterion are discarded. If you are getting too many sequences or uninteresting sequences, try increasing this setting. If you are getting too few sequences, try decreasing this setting.

**Maximum sequence size**. You can set the maximum number of distinct *item sets* (as opposed to *items*) in a sequence. If the sequences of interest are relatively short, you can decrease this setting to speed up building the sequence set.

**Predictions to add to stream**. Specify the number of predictions to be added to the stream by the resulting generated Model node. For more information, see “Generated Sequence Rules Node” in Chapter 12 on page 454.

**Sequence Node Expert Options**

For those with detailed knowledge of the Sequence node's operation, the following expert options allow you to fine-tune the model-building process. To access expert options, set the Mode to Expert on the Expert tab.
**Set maximum duration.** If selected, sequences will be limited to those with a duration (the time between the first and last item set) less than or equal to the value specified. If you haven't specified a time field, the duration is expressed in terms of rows (records) in the raw data. If the Time field used is a time, date or timestamp field, the duration is expressed in seconds. For numeric fields, the duration is expressed in the same units as the field itself.

**Set pruning value.** The CARMA algorithm used in the Sequence node periodically removes (prunes) infrequent item sets from its list of potential item sets during processing to conserve memory. Select this option to adjust the frequency of pruning. The number specified determines the frequency of pruning. Enter a smaller value to decrease the memory requirements of the algorithm (but potentially increase the training time required), or enter a larger value to speed up training (but potentially increase memory requirements).

**Set maximum sequences in memory.** If selected, the CARMA algorithm will limit its memory store of candidate sequences during model building to the number of sequences specified. Select this option if Clementine is using too much memory during the building of Sequence models. Note that the maximum sequences value you specify here is the number of candidate sequences tracked internally as the
model is built. This number should be much larger than the number of sequences you expect in the final model.

For example, consider this list of transactions:

<table>
<thead>
<tr>
<th>ID</th>
<th>Time</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>1</td>
<td>apples</td>
</tr>
<tr>
<td>1001</td>
<td>2</td>
<td>bread</td>
</tr>
<tr>
<td>1001</td>
<td>3</td>
<td>cheese</td>
</tr>
<tr>
<td>1001</td>
<td>4</td>
<td>dressing</td>
</tr>
</tbody>
</table>

If you build a model on these data with the time stamp tolerance set to 2, you would get the usual singleton item sets of apples, bread, cheese, and dressing, but you would also get compound item sets of apples & bread, bread & cheese, cheese & dressing, apples & bread & cheese, and bread & cheese & dressing. Note that you would not get the item set apples & bread & cheese & dressing, because dressing does not occur within two time units of apples, even though it does occur within two time units of cheese.

Item sets formed in this way will be considered to have a compound time stamp, with both a start time and an end time. Note that for such item sets, the time between the start time and end time will always be less than or equal to the tolerance value.

If unselected, only items with the same time stamp and ID values will be considered part of the same item set.

**Constrain gaps between item sets.** This option allows you to specify constraints on the time gaps that separate item sets. If selected, item sets with time gaps smaller than the Minimum gap or larger than the Maximum gap that you specify will not be considered to form part of a sequence. Use this option to avoid counting sequences that include long time intervals or those that take place in a very short time span.

*Note:* If the Time field used is a time, date or timestamp field, the time gap is expressed in seconds. For numeric fields, the time gap is expressed in the same units as the time field.
For example, consider this list of transactions:

<table>
<thead>
<tr>
<th>ID</th>
<th>Time</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>1</td>
<td>apples</td>
</tr>
<tr>
<td>1001</td>
<td>2</td>
<td>bread</td>
</tr>
<tr>
<td>1001</td>
<td>5</td>
<td>cheese</td>
</tr>
<tr>
<td>1001</td>
<td>6</td>
<td>dressing</td>
</tr>
</tbody>
</table>

If you build a model on these data with the minimum gap set to 2, you would get the following sequences:

apples => cheese

apples => dressing

bread => cheese

bread => dressing

You would not see sequences such as apples => bread, because the gap between apples and bread is smaller than the minimum gap. Similarly, if the data were instead:

<table>
<thead>
<tr>
<th>ID</th>
<th>Time</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>1</td>
<td>apples</td>
</tr>
<tr>
<td>1001</td>
<td>2</td>
<td>bread</td>
</tr>
<tr>
<td>1001</td>
<td>5</td>
<td>cheese</td>
</tr>
<tr>
<td>1001</td>
<td>20</td>
<td>dressing</td>
</tr>
</tbody>
</table>

and the maximum gap were set to 10, you would not see any sequences with dressing, because the gap between cheese and dressing is too large for them to be considered part of the same sequence.
Chapter 12

Generated Models

Overview of Generated Models

Generated models are the fruits of your data modeling labor. A generated model node is created whenever you successfully execute a modeling node. Generated models contain information about the model created and provide a mechanism for using that model to generate predictions and facilitate further data mining.

Generated models are placed in the generated models palette (located on the Models tab in the managers window in the upper right corner of the Clementine window) when they are created. From there they can be selected and browsed to view details of the model. Generated models other than unrefined rule models can be placed into the stream to generate predictions or to allow further analysis of their properties.

You can identify the type of a generated model node from its icon:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Node type</th>
<th>Icon</th>
<th>Node type</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Neural Network Icon" /></td>
<td>Neural Network</td>
<td><img src="image2" alt="Kohonen Net Icon" /></td>
<td>Kohonen Net</td>
</tr>
<tr>
<td><img src="image3" alt="C5.0 Tree model Icon" /></td>
<td>C5.0 Tree model</td>
<td><img src="image4" alt="Linear Regression Equation Icon" /></td>
<td>Linear Regression Equation</td>
</tr>
<tr>
<td><img src="image5" alt="Ruleset Icon" /></td>
<td>Ruleset</td>
<td><img src="image6" alt="K-Means model Icon" /></td>
<td>K-Means model</td>
</tr>
<tr>
<td><img src="image7" alt="Logistic Regression Equation Icon" /></td>
<td>Logistic Regression Equation</td>
<td><img src="image8" alt="C&amp;R Tree model Icon" /></td>
<td>C&amp;R Tree model</td>
</tr>
</tbody>
</table>
Chapter 12

<table>
<thead>
<tr>
<th>Icon</th>
<th>Node type</th>
<th>Icon</th>
<th>Node type</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Factor/PCA Equation Icon]</td>
<td>Factor/PCA Equation</td>
<td>![Sequence set Icon]</td>
<td>Sequence set</td>
</tr>
<tr>
<td>![Unrefined association rules Icon]</td>
<td>Unrefined association rules (generated models palette only)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following topics provide information on using generated models in Clementine. For an in-depth understanding of the algorithms used in Clementine, consult the *Clementine Algorithms Guide* available on the product CD.

**Working with Generated Models in the Generated Models Palette**

The generated models palette (on the Models tab in the managers window) allows you to use, examine, and modify generated model nodes in various ways. Two context menus provide access to these features.

**Figure 12-1**
*Generated model context menu*

![Generated model context menu]

Right-clicking directly on a generated model node in the generated models palette opens a context menu with the following options for modifying the node:

- **Add to Stream.** Adds the generated model node to the currently active stream. If there is a selected node in the stream, the generated model node will be connected to the selected node when such a connection is possible.
- **Browse.** Opens the model browser for the node.
- **Rename and Annotate.** Allows you to rename the generated model node and/or modify the annotation for the node.
- **Save.** Saves the node to an external file.
- **Export PMML.** Exports the model as predictive model markup language (PMML), which can be used with SPSS SmartScore for scoring new data outside of Clementine. Export PMML is available for all generated model nodes, except those created by CEMI modeling nodes. For more information, see “Exporting Models” in Chapter 13 on page 464.
- **Export C code.** Exports the model as C code that can be used to score new data outside of Clementine. This option is available for generated Net, C5.0 Tree, generated Kohonen, and generated K-Means nodes. For more information, see “Exporting Models” in Chapter 13 on page 464.
- **Add to Project.** Saves the generated model and adds it to the current project. On the Classes tab, the node will be added to the Generated Models folder. On the CRISP-DM tab, it will be added to the default project phase. (See Setting the Default Project Phase for information on how to change the default project phase.)
- **Delete.** Deletes the node from the palette.

**Figure 12-2**
*Generated models palette context menu*

Right-clicking on an unoccupied area in the generated models palette opens a context menu with the following options:
- **Open Model.** Loads a generated model previously created in Clementine.
- **Load Palette.** Loads a saved palette from an external file.
- **Save Palette.** Saves the entire contents of the generated models palette to an external file.
- **Clear Palette.** Deletes all nodes from the palette.
Chapter 12

- **Add to Project.** Saves the generated models palette and adds it to the current project. On the Classes tab, the node will be added to the Generated Models folder. On the CRISP-DM tab, it will be added to the default project phase.

- **Import PMML.** Loads a model from an external file. You can open, browse, and score PMML models created by SPSS and AnswerTree.

**Importing Models Saved as PMML**

Models exported as PMML from Clementine or another application, such as SPSS or AnswerTree, can easily be brought into the generated models palette of Clementine.

**Figure 12-3**
*Selecting the XML file for a model saved using PMML*
**Use variable labels.** The PMML may specify both variable names and variable labels (such as Referrer ID for RefID) for variables in the data dictionary. Select this option to use variable labels, if they are present in the originally exported PMML.

**Use value labels.** The PMML may specify both values and value labels (such as Male for M or Female for F) for a variable. Select this option to use the value labels, if they are present in the PMML.

If you have selected the above label options but there are no variable or value labels in the PMML, then the variable names and literal values are used as normal. By default, both options are selected.

**Supported Model Types**

The following PMML models can be browsed and scored in Clementine:

- CHAID
- Exhaustive CHAID
- C&RT
- QUEST
- Logistic regression (multinomial and conditional logistic)
- TwoStep cluster

**Using Generated Models in Streams**

The generated models can be placed in streams to score new data and generate new nodes. **Scoring** data allows you to use the information gained from model building to create predictions for new records. For some models, generated model nodes can also give you additional information about the quality of the prediction, such as confidence values or distances from cluster centers. Generating new nodes allows you to easily create new nodes based on the structure of the generated model. For example, most models that perform input field selection allow you to generate Filter nodes that will pass only input fields that the model identified as important.

**To use a generated model node for scoring data:**

- Select the desired model by clicking it in the generated models palette.
Add the model to the stream by clicking the desired location in the stream canvas.

Connect the generated model node to a data source or stream that will pass data to it.

Add or connect one or more processing or output nodes (such as a Table node) to the generated model node.

Execute one of the nodes downstream from the generated model node.

Note that you cannot use Unrefined Rule nodes for scoring data. To score data based on an association rule model, use the Unrefined Rule node to generate a Ruleset node, and use the Ruleset node for scoring. For more information, see “Generating a Ruleset” on page 411.

To use a generated model node for generating processing nodes:

Browse (on the palette) or edit (on the stream canvas) the model.

Select the desired node type from the Generate menu of the generated model browser window. The options available will vary depending on the type of generated model node. See the specific generated model type for details about what you can generate from a particular model.

Using the Generated Model Browsers

The generated model browsers allow you to examine and use the results of your models. From the browser, you can save, print, or export the generated model, examine the model summary, and view or edit annotations for the model. For some types of generated models, you can also generate new nodes, such as Filter nodes or Ruleset nodes. For some models, you can also view model parameters, such as rules or cluster centers. For some types of models (tree-based models and cluster models), you can view a graphical representation of the structure of the model. Controls for using the generated model browsers are described below.
Menus

File Menu. All generated models have a File menu, containing the following options:

- **Save Node.** Saves the generated model node to a file.
- **Close.** Closes the current generated model browser.
- **Header and Footer.** Allows you to edit the page header and footer for printing from the node.
- **Page Setup.** Allows you to change the page setup for printing from the node.
- **Print Preview.** Displays a preview of how the node will look when printed. Select the information you want to preview from the submenu.
- **Print.** Prints the contents of the node. Select the information you want to print from the submenu.
- **Export Text.** Exports the contents of the node to a text file. Select the information you want to export from the submenu.
- **Export HTML.** Exports the contents of the node to an HTML file. Select the information you want to export from the submenu.
- **Export PMML.** Exports the model as predictive model markup language (PMML), which can be used with other PMML-compatible software.
- **Export C code.** Exports the model as C code, which can be compiled and used with other applications.

Generate menu. Most generated models also have a Generate menu, allowing you to generate new nodes based on the generated model. The options available from this menu will depend on the type of model you are browsing. See the specific generated model type for details about what you can generate from a particular model.

Tabs

Generated models information appears on several tabs, to make the information more manageable. The set of tabs available varies depending on the type of generated model.

- **Model.** This tab contains basic model information, such as cluster centers (for cluster models) or rules (for rule sets).
- **Viewer.** For tree-based models (C5.0 and C&R Tree) and cluster models (K-Means and Two Step), the Viewer tab provides a graphical representation model results. For trees, the Viewer tab shows split criteria and the distribution of target values in each branch of the tree. For clusters, the Viewer tab shows the mean and distribution of values for each cluster, enabling you to visually compare clusters.

- **Advanced.** For linear regression, logistic regression, and factor/PCA models, the Advanced tab displays detailed statistical analysis of the model.

- **Summary.** The Summary tab contains information about model performance, fields used, build settings, and training summary.

- **Annotations.** The Annotations tab contains annotations about the generated model. For more information, see “Annotating Nodes” in Chapter 4 on page 70.

### Generated Net Node

Generated Net nodes represent the neural networks created by Neural Net nodes. They contain all of the information captured by the trained network, as well as information about the neural network's characteristics, such as accuracy and architecture.

To see information about the neural network model, right-click the generated Net node and select **Browse** from the context menu (or **Edit** for nodes in a stream).

You can add the network model to your stream by selecting the icon in the generated models palette and then clicking the stream canvas where you want to place the node, or by right-clicking the icon and selecting **Add to Stream** from the context menu. Then connect your stream to the node, and you are ready to pass data to the network model to generate predictions. The data coming into the generated model node must contain the same input fields, with the same types, as the training data used to create the model. (If fields are missing or field types are mismatched, you will see an error message when you execute the stream.)

When you execute a stream containing a generated Net node, the Net node adds a new field for each output field from the original training data. The new field contains the network's prediction for the corresponding output field. The name of each new prediction field is the name of the output field being predicted, with \$N- added to the beginning. For example, for an output field named *profit*, the predicted values would appear in a new field called \$N-profit. For symbolic output fields, a second new field is also added, containing the confidence for the prediction. The confidence field is named in a similar manner, with \$NC- added to the beginning of the original output field name. In a stream with multiple generated Net nodes in a series predicting the
same output field(s), the new predicted and confidence field names will include numbers to distinguish them from each other. The first Net node in the stream will use the usual names, the second node will use names starting with $N1$- and $NC1$-, the third node will use names starting with $N2$- and $NC2$-, and so on.

**Confidence for neural networks.** Confidence values for neural networks are provided for symbolic output fields and are computed as follows:

- **Flag data.** Confidence is computed as $\text{abs}(0.5 – \text{Raw Output}) \times 2$. Values are converted into a scale of 0 to 1. If the output unit value is below 0.5, it is predicted as 0 (false), and if it is 0.5 or above, it is predicted as 1 (true). For example, if the Neural Net prediction value is 0.72, then this is displayed as “true” and the confidence will be $(0.5 – 0.72) \times 2 = 0.64$.

- **Set data.** Set output fields are internally converted to flags for neural networks, so there is a separate raw output value for each category of the output field. Values are converted into a scale of 0 to 1. Confidence is computed as $(\text{Highest Raw Output} – \text{Second Highest Raw Output})$. The highest scaled value defines which predicted set value is chosen, and the difference between the highest scaled value and the second highest scaled value is the confidence. For example, if there are four set values (*red*, *blue*, *white*, *black*) and the scaled values produced by Neural Net are $\text{red} = 0.32$, $\text{blue} = 0.85$, $\text{white} = 0.04$, and $\text{black} = 0.27$, then the predicted set value would be *blue*, and the confidence would be $0.85 – 0.32 = 0.53$.

**Generating a Filter node.** The Generate menu allows you to create a new Filter node to pass input fields based on the results of the model. For more information, see “Generating a Filter Node from a Neural Network” on page 392.

**Generated Neural Network Summary Tab**

On the generated Net node Summary tab, you will see information about the network itself (Analysis), fields used in the network (Fields), settings used when building the model (Build Settings), and model training (Training Summary). You can also perform file operations, including printing, saving, and exporting, from the File menu, and you can generate new Filter nodes from the Generate menu.

When you first browse a generated Net node, the Summary tab results may be collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or use the Expand All button to show all results. To hide results
when finished viewing them, use the expander control to collapse the specific results you want to hide, or use the Collapse All button to collapse all results.

Figure 12-4
Sample generated Net node Summary tab

Analysis. The analysis section displays information about the estimated accuracy of the network, the architecture or topology of the network, and the relative importance of fields, as determined by sensitivity analysis (if you requested it). If you have executed an Analysis node attached to this modeling node, information from that analysis will also appear in this section. For more information, see “Analysis Node” in Chapter 14 on page 504.

- Estimated accuracy. This is an index of the accuracy of the predictions. For symbolic outputs, this is simply the percentage of records for which the predicted value is correct. For numeric targets, the calculation is based on the differences between the predicted values and the actual values in the training data. The formula for finding the accuracy for numeric fields is
where Actual is the actual value of the output field, Predicted is the value predicted by the network, and Range of Output Field is the range of values for the output field (the highest value for the field minus the lowest value). This accuracy is calculated for each record, and the overall accuracy is the average of the values for all records in the training data.

Because these estimates are based on the training data, they are likely to be somewhat optimistic. The accuracy of the model on new data will usually be somewhat lower than this.

- **Architecture.** For each layer in the network, the number of units in that layer is listed.

- **Relative Importance of Inputs.** This section contains the results of the sensitivity analysis if you requested one. The input fields are listed in order of importance, from most important to least important. The value listed for each input is a measure of its relative importance, varying between 0 (a field that has no effect on the prediction) and 1.0 (a field that completely determines the prediction).

**Fields.** This section lists the fields used as target(s) and inputs in building the model.

**Build Settings.** This section contains information on the settings used in building the model.

**Training Summary.** This section shows the type of the model, the stream used to create it, the user who created it, when it was built, and the elapsed time for building the model.
Generating a Filter Node from a Neural Network

You can generate a Filter node from a generated neural network model. The dialog box contains a list of fields in descending order of relative importance in the model. Select the fields to be retained in the model, and click OK. The generated Filter node will appear on the stream canvas.

Selecting fields. Click on the last field you want to retain (the one with the smallest relative importance that meets your criteria). This will select that field and all fields with a higher relative importance. The top field (with the highest importance) is always selected.

Logistic Regression Equation Node

Logistic Regression Equation nodes represent the equations estimated by Logistic Regression nodes. They contain all of the information captured by the logistic regression model, as well as information about the model structure and performance.

To see information about the logistic regression model, right-click the Logistic Regression Equation node and select Browse from the context menu (or Edit for nodes in a stream).

You can add the logistic regression model to your stream by clicking the icon in the generated models palette and then clicking the stream canvas where you want to place the node, or by right-clicking the icon and selecting Add to Stream from the context menu. Then connect your stream to the node, and you are ready to pass data to the logistic regression model to generate predictions. The data coming into the generated model node must contain the same input fields, with the same types, as
the training data used to create the model. (If fields are missing or field types are mismatched, you will see an error message when you execute the stream.)

When you execute a stream containing a Logistic Regression Equation node, the node adds two new fields containing the model's prediction and the associated probability. The names of the new fields are derived from the name of the output field being predicted, prefixed with $L-$ for the predicted category and $LP-$ for the associated probability. For example, for an output field named `colorpref`, the new fields would be named `$L-colorpref` and `$LP-colorpref`. In addition, if you have selected the Append all probabilities expert option in the Logistic Regression node, an additional field will be added for each category of the output field, containing the probability belonging to the corresponding category for each record. These additional fields are named based on the values of the output field, prefixed by `$LP-`. For example, if the legal values of `colorpref` are `Red`, `Green`, and `Blue`, three new fields will be added: `$LP-Red`, `$LP-Green`, and `$LP-Blue`. In a stream with multiple Logistic Regression Equation nodes in a series predicting the same output field, the new predicted and confidence field names will include numbers to distinguish them from each other. The first Logistic Regression Equation node in the stream will use the usual names, the second node will use names starting with `$L1-$` and `$LP1-$`, the third node will use names starting with `$L2-$` and `$LP2-$`, and so on.

**Generating a Filter node.** The Generate menu allows you to create a new Filter node to pass input fields based on the results of the model. Fields that are dropped from the model due to multicollinearity will be filtered by the generated node, as well as fields not used in the model.

**Logistic Regression Equation Model Tab**

On the Logistic Regression Equation node Model tab, you will see the actual equations estimated by the Logistic Regression node; one equation for each category in the target field except the baseline category. The equations are displayed in a tree format.

When you first browse a Logistic Regression Equation node, the Model tab results start out collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or click the Expand All button to show all results. To hide results when finished viewing them, use the expander control to collapse the specific results you want to hide, or click the Collapse All button to collapse all results.
Equation for. Shows the regression equations used to derive the target category probabilities, given a set of predictor values. The last category of the target field is considered the baseline category; the equations shown give the log-odds for the other target categories relative to the baseline category for a particular set of predictor values. The predicted probability for each category for the given predictor pattern is derived from these log-odds values.

How Are Probabilities Calculated?

Each equation calculates the log-odds for a particular target category, relative to the baseline category. The log-odds, also called the logit, is the ratio of the probability for specified target category to that of the baseline category, with the natural logarithm function applied to the result. For the baseline category, the odds of the category relative to itself is 1.0, and thus the log-odds are 0. You can think of this as an implicit equation for the baseline category where all coefficients are zero.
To derive the probability from the log-odds for a particular target category, you take the logit value calculated by the equation for that category and apply the following formula:

\[ P(\text{group}_i) = \frac{\exp(g_i)}{\sum_k \exp(g_k)} \]

where \( g \) is the calculated log-odds, \( i \) is the category index, and \( k \) goes from 1 to the number of target categories.

**Logistic Regression Equation Summary Tab**

On the Logistic Regression Equation node Summary tab, you will see information about the model itself (Analysis), fields used in the model (Fields), settings used when building the model (Build Settings), and model training (Training Summary).
**Analysis.** If you have executed an Analysis node attached to this modeling node, information from that analysis will also appear in this section. For more information, see “Analysis Node” in Chapter 14 on page 504.

**Fields.** This section lists the fields used as the target and the inputs in building the model.

**Build Settings.** This section contains information on the settings used in building the model.

**Training Summary.** This section shows the type of the model, the stream used to create it, the user who created it, when it was built, and the elapsed time for building the model.
Logistic Regression Advanced Output

The advanced output for logistic regression (also known as **nominal regression**) gives detailed information about the estimated model and its performance. Most of the information contained in the advanced output is quite technical, and extensive knowledge of logistic regression analysis is required to properly interpret this output.

**Warnings.** Indicates any warnings or potential problems with the results.

**Step Summary.** Lists changes to the model at each step when using automatic field selection (the Stepwise, Forwards, Backwards, or Backwards Stepwise methods).

**Case processing summary.** Lists the number of records processed, broken down by each symbolic field in the model.
**Model fitting information.** Shows the likelihood ratio test of your model (Final) against one in which all of the parameter coefficients are 0 (Intercept Only).

**Goodness-of-fit chi-square statistics (optional).** Shows Pearson's and likelihood-ratio chi-square statistics. These statistics test the overall fit of the model to the training data.

**Pseudo R-square (optional).** Shows the Cox and Snell, Nagelkerke, and McFadden $R^2$-square measures of model fit.

**Likelihood ratio tests (optional).** Shows statistics testing whether the coefficients of the model effects are statistically different from 0.

**Parameter estimates (optional).** Shows estimates of the equation coefficients, tests of those coefficients, odds ratios derived from the coefficients (labeled $\exp(B)$), and confidence intervals for the odds ratios.

**Asymptotic covariance/correlation matrix (optional).** Shows the asymptotic covariances and/or correlations of the parameter estimates.

**Classification (optional).** Shows the matrix of predicted and actual output field values with percentages.

**Observed and predicted frequencies (optional).** For each covariate pattern, shows the observed and predicted frequencies for each output field value. This table can be quite large, especially for models with numeric input fields. If the resulting table would be too large to be practical, it is omitted, and a warning appears.

---

**Linear Regression Equation Node**

Linear Regression Equation nodes represent the equations estimated by Linear Regression nodes. They contain all of the information captured by the linear regression model, as well as information about the model structure and performance.

To see information about the logistic regression model, right-click the Linear Regression Equation node and select Browse from the context menu (or Edit for nodes in a stream).

You can add the linear regression model to your stream by clicking the icon in the generated models palette and then clicking the stream canvas where you want to place the node, or by right-clicking the icon and selecting Add to Stream from the context menu. Then connect your stream to the node, and you are ready to pass data to the
linear regression model to generate predictions. The data coming into the generated model node must contain the same input fields, with the same types, as the training data used to create the model. (If fields are missing or field types are mismatched, you will see an error message when you execute the stream.)

When you execute a stream containing a Linear Regression Equation node, the node adds a new field containing the model's prediction for the output field. The name of the new field is derived from the name of the output field being predicted, prefixed with $E$. For example, for an output field named profit, the new field would be named $E$-profit. In a stream with multiple Linear Regression Equation nodes in a series predicting the same output field, the new predicted and confidence field names will include numbers to distinguish them from each other. The first Linear Regression Equation node in the stream will use the usual name, the second node will use a name starting with $E1$, the third node will use a name starting with $E2$, and so on.

**Generating a Filter node.** The Generate menu allows you to create a new Filter node to pass input fields based on the results of the model. This is most useful with models built using one of the field selection methods. For more information, see “Linear Regression Node Model Options” in Chapter 11 on page 334.

You can assess the linear regression model by placing the Linear Regression Equation node in the stream and using various graph and output nodes to examine its predictions. For example, attaching an Analysis node gives you information on how well the predicted values match the actual values. You can also use a Plot node to display predicted values versus actual values, which can help you to identify the records that are most difficult for the model to classify accurately and to identify systematic errors in the model.

You can also assess the linear regression model using the information available in the advanced output. To view the advanced output, select the Advanced tab of the generated model browser. The advanced output contains a lot of detailed information and is meant for users with extensive knowledge of linear regression. For more information, see “Linear Regression Equation Advanced Output” on page 401.

**Linear Regression Equation Summary Tab**

On the Linear Regression Equation node Summary tab, you will see information about the model itself (Analysis), fields used in the model (Fields), settings used when building the model (Build Settings), and model training (Training Summary).
When you first browse a Linear Regression Equation node, the Summary tab results start out collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or click the Expand All button to show all results. To hide results when finished viewing them, use the expander control to collapse the specific results you want to hide, or click the Collapse All button to collapse all results.

**Figure 12-9**
Sample Linear Regression Equation node Summary tab

<table>
<thead>
<tr>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>farmvalue * 1.436 +</td>
</tr>
<tr>
<td>farmsize + 232.4 +</td>
</tr>
<tr>
<td>landquality* 45,876.6 +</td>
</tr>
<tr>
<td>rainfall * 3,940.3 +</td>
</tr>
<tr>
<td>-555,037.5</td>
</tr>
</tbody>
</table>

**Fields.** The analysis section displays each input field with its coefficient in the regression equation. The complete regression equation is the sum of all entries. If you have executed an Analysis node attached to this modeling node, information from that analysis will also appear in this section. For more information, see “Analysis Node” in Chapter 14 on page 504.

**Fields.** This section lists the fields used as the target and the inputs in building the model.


**Build Settings.** This section contains information on the settings used in building the model.

**Training Summary.** This section shows the type of the model, the stream used to create it, the user who created it, when it was built, and the elapsed time for building the model.

### Linear Regression Equation Advanced Output

Figure 12-10
Sample Linear Regression Equation node advanced output

The advanced output for linear regression gives detailed information on the estimated model and its performance. Most of the information contained in the advanced output is quite technical, and extensive knowledge of linear regression analysis is required to properly interpret this output.
**Warnings.** Indicates any warnings or potential problems with the results.

**Descriptive statistics (optional).** Shows the number of valid records (cases), the mean, and the standard deviation for each field in the analysis.

**Correlations (optional).** Shows the correlation matrix of input and output fields. One-tailed significance and the number of records (cases) for each correlation are also displayed.

**Variables entered/removed.** Shows fields added to or removed from the model at each step for Stepwise, Forwards, and Backwards regression methods. For the Enter method, only one row is shown entering all fields immediately.

**Model summary.** Shows various summaries of model fit. If the R-Squared Change option is selected in the Linear Regression node, change in model fit is reported at each step for Stepwise, Forwards, and Backwards methods. If the Selection Criteria option is selected in the Linear Regression node, additional model fit statistics are reported at each step, including Akaike Information Criterion, Amemiya Prediction Criterion, Mallows' Prediction Criterion, and Schwarz Bayesian Criterion.

**ANOVA.** Shows the analysis of variance (ANOVA) table for the model.

**Coefficients.** Shows the coefficients of the model and statistical tests of those coefficients. If the Confidence interval option is selected in the Linear Regression node, 95% confidence intervals are also reported in this table. If the Part and partial correlations option is selected, part and partial correlations are also reported in this table. Finally, if the Collinearity Diagnostics option is selected, collinearity statistics for input fields are reported in this table.

**Coefficient correlations (optional).** Shows correlations among coefficient estimates.

**Collinearity diagnostics (optional).** Shows collinearity diagnostics for identifying situations in which the input fields form a linearly dependent set.

**Casewise diagnostics (optional).** Shows the records with the largest prediction errors.

**Residuals statistics (optional).** Shows summary statistics describing the distribution of prediction errors.
**Factor Equation Node**

Factor Equation nodes represent the factor analysis and principal component analysis (PCA) models created by Factor/PCA nodes. They contain all of the information captured by the trained model, as well as information about the model’s performance and characteristics.

To see information about the factor/PCA model, right-click the Factor Equation node and select Browse from the context menu (or Edit for nodes in a stream).

You can add the Factor Equation node to your stream by clicking the icon in the generated models palette and then clicking the stream canvas where you want to place the node, or by right-clicking the icon and selecting Add to Stream from the context menu. Then connect your stream to the node, and you are ready to pass data to the factor model to compute factor or component scores. The data coming into the generated model node must contain the same input fields, with the same types, as the training data used to create the model. (If fields are missing or field types are mismatched, you will see an error message when you execute the stream.)

When you execute a stream containing a Factor Equation node, the Factor Equation node adds a new field for each factor or component in the model. The new field names are derived from the model name, prefixed by $F-$ and suffixed by -$n$, where $n$ is the number of the factor or component. For example, if your model is named Factor and contains three factors, the new fields would be named $F$-Factor-1, $F$-Factor-2, and $F$-Factor-3. In a stream with multiple Factor Equation nodes in a series predicting the same output field(s), the new field names will include numbers in the prefix to distinguish them from each other. The first Factor Equation node in the stream will use the usual names, the second node will use names starting with $F1-$, the third node will use names starting with $F2-$, and so on.

To get a better sense of what the factor model has encoded, you can do some more downstream analysis. A useful way to view the result of the factor model is to view the correlations between factors and input fields using a Statistics node. This shows you which input fields load heavily on which factors and can help you discover if your factors have any underlying meaning or interpretation. For more information, see “Statistics Node” in Chapter 14 on page 518.

You can also assess the factor model using the information available in the advanced output. To view the advanced output, select the Advanced tab of the generated model browser. The advanced output contains a lot of detailed information and is meant for users with extensive knowledge of factor analysis or PCA. For more information, see “Factor Equation Advanced Output” on page 406.
Chapter 12

Factor Equation Model Tab

On the Factor Equation node Model tab, you will see information about how the scores are calculated.

Figure 12-11
Sample Factor Equation node Model tab

Equation for Factor-1

\[0.0006907 \times \text{farmsize} +
+ 0.01114 \times \text{rainfall} +
+ 0.1405 \times \text{landquality} +
+ 0.00004731 \times \text{claimvalue} +
+ -3.127\]

Equation for Factor-2

\[0.0009473 \times \text{farmsize} +
+ -0.02746 \times \text{rainfall} +
+ 0.215 \times \text{landquality} +
+ -0.0000003137 \times \text{claimvalue} +
+ -0.5987\]

Equation for. Shows the factor score equation for each factor. Factor or component scores are calculated by multiplying each input field value by its coefficient and summing the results.
**Factor Equation Summary Tab**

On the Factor Equation node Summary tab, you will see information about the model itself (Analysis), fields used in the model (Fields), settings used when building the model (Build Settings), and model training (Training Summary).

When you first browse a Factor Equation node, the Summary tab results start out collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or click the Expand All button to show all results. To hide results when finished viewing them, use the expander control to collapse the specific results you want to hide, or click the Collapse All button to collapse all results.

**Figure 12-12**
*Sample Factor Equation node Summary tab*

**Analysis.** The analysis section displays the number of factors retained in the factor/PCA model.
**Fields.** This section lists the fields used as inputs in building the model.

**Build Settings.** This section contains information on the settings used in building the model.

**Training Summary.** This section shows the type of the model, the stream used to create it, the user who created it, when it was built, and the elapsed time for building the model.

---

**Factor Equation Advanced Output**

Figure 12-13
Sample Factor Equation node advanced output

![Factor Analysis Table](image)

<table>
<thead>
<tr>
<th>Component</th>
<th>Initial Eigenvalues</th>
<th>Extraction Sums of Squared Loadings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>% of Variance</td>
</tr>
<tr>
<td>1</td>
<td>1.833</td>
<td>45.816</td>
</tr>
<tr>
<td>2</td>
<td>1.049</td>
<td>26.236</td>
</tr>
<tr>
<td>3</td>
<td>.954</td>
<td>23.860</td>
</tr>
<tr>
<td>4</td>
<td>.164</td>
<td>4.088</td>
</tr>
</tbody>
</table>
The advanced output for factor analysis gives detailed information on the estimated model and its performance. Most of the information contained in the advanced output is quite technical, and extensive knowledge of factor analysis is required to properly interpret this output.

**Warnings.** Indicates any warnings or potential problems with the results.

**Communalities.** Shows the proportion of each field's variance that is accounted for by the factors or components. *Initial* gives the initial communalities with the full set of factors (the model starts with as many factors as input fields), and *Extraction* gives the communalities based on the retained set of factors.

**Total variance explained.** Shows the total variance explained by the factors in the model. *Initial Eigenvalues* shows the variance explained by the full set of initial factors. *Extraction Sums of Squared Loadings* shows the variance explained by factors retained in the model. *Rotation Sums of Squared Loadings* shows the variance explained by the rotated factors. Note that for oblique rotations, *Rotation Sums of Squared Loadings* shows only the sums of squared loadings and does not show variance percentages.

**Factor (or component) matrix.** Shows correlations between input fields and unrotated factors.

**Rotated factor (or component) matrix.** Shows correlations between input fields and rotated factors for orthogonal rotations.

**Pattern matrix.** Shows the partial correlations between input fields and rotated factors for oblique rotations.

**Structure matrix.** Shows the simple correlations between input fields and rotated factors for oblique rotations.

**Factor correlation matrix.** Shows correlations among factors for oblique rotations.

### Unrefined Rule Model

Unrefined Rule models represent the rules discovered by one of the association rule modeling nodes (Apriori or GRI). These models contain information about the rules extracted from the data. Unrefined Rule models are not designed for generating predictions directly, and they cannot be added to streams.
To see information about the Unrefined Rule model, right-click the model and select Browse from the context menu.

**Generating nodes.** The Generate menu allows you to create new nodes based on the rules.

- **Select Node.** Generates a Select node to select records to which the currently selected rule applies. This option is disabled if no rule is selected.
- **Rule set.** Generates a Ruleset node to predict values for a single target field. For more information, see “Generating a Ruleset” on page 411.

**Unrefined Rules Model Tab**

On the Model tab of an Unrefined Rule model, you will see a table containing the rules extracted by the algorithm. Each row in the table represents a rule. The first column represents the consequent (the “then” part of the rule), and subsequent columns represent the antecedents (the “if” part of the rule).

![Sample Unrefined Rule node Model tab](image)
Each rule is shown in the following format:

\[
\text{Consequent} \quad \text{Antecedent 1} \quad \text{Antecedent 2}
\]

\[
\text{Drug} = \text{drugY} \quad \text{Sex} = \text{F} \quad \text{BP} = \text{HIGH}
\]

The example rule is interpreted as *for records where Sex = “F” and BP = “HIGH”, Drug is likely to be drugY.* If you select Show Instances/Confidence from the toolbar, each rule will also show information on the number of records to which the rule applies—that is, for which the antecedents are true (Instances), the proportion of the training data represented by the instances (Support), and the proportion of those records for which the entire rule, antecedents, and consequent is true (Confidence).

**Sort menu.** The Sort menu controls sorting of the rules. Direction of sorting (ascending or descending) can be changed using the sort direction button on the toolbar. Select from the following sort keys:

- **Support * Confidence.** Sorts rules by the product of support (as defined above) and confidence. This emphasizes rules that are both accurate and apply to a large proportion of the training data. This is the default.
- **Consequent.** Sorts rules alphabetically by the predicted value (the consequent).
- **Number of Antecedents.** Sorts rules by the number of antecedents (rule length).
- **Support.** Sorts rules by support.
- **Confidence.** Sorts rules by confidence.
- **Length.** Sorts rules by length or number of antecedents.
- **Lift.** Sorts rules by lift, indicating an improvement in expected return over that expected without a classifier or model. The lift statistic is defined as $P(\text{Consequent} | \text{Antecedents}) / P(\text{Consequent})$.

**Unrefined Rule Summary Tab**

On the Summary tab of an Unrefined Rule model, you will see information about the model itself (Analysis), fields used in the model (Fields), settings used when building the model (Build Settings), and model training (Training Summary).

When you first browse an Unrefined Rule model, the Summary tab results start out collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or click the Expand All button to show all results. To hide results
Chapter 12

When finished viewing them, use the expander control to collapse the specific results you want to hide, or click the Collapse All button to collapse all results.

**Figure 12-15**
Sample Unrefined Rule node Summary tab

**Analysis.** The analysis section displays the number of rules discovered and the minimum and maximum for support, lift, and confidence of rules in the ruleset.

**Fields.** This section lists the fields used as consequents and antecedents in building the model.

**Build Settings.** This section contains information on the settings used in building the model.
Training Summary. This section shows the type of the model, the stream used to create it, the user who created it, when it was built, and the elapsed time for building the model.

Generating a Ruleset

Figure 12-16
Generate Ruleset dialog box

To generate a Ruleset from an Unrefined Rule model or Sequence Ruleset node, select Rule set from the Generate menu in the generated model browser. You can specify the following options for translating the rules into a ruleset:

Rule set name. Allows you to specify the name of the new generated Ruleset node.

Create node on. Controls the location of the new generated Ruleset node. Select Canvas, GM Palette, or Both.

Target field. Determines which output field will be used for the generated Ruleset node. Select a single output field from the list.

Minimum support. Specify the minimum support for rules to be preserved in the generated ruleset. Rules with support less than the specified value will not appear in the new ruleset.

Minimum confidence. Specify the minimum confidence for rules to be preserved in the generated ruleset. Rules with confidence less than the specified value will not appear in the new ruleset.

Default value. Allows you to specify a default value for the target field that is assigned to scored records for which no rule fires.
Generated Ruleset Node

Generated Ruleset nodes represent the rules for predicting a particular output field discovered by one of the association rule modeling nodes (Apriori or GRI), by the Build C5.0 node, or by the C&R Tree node. For association rules, the generated Ruleset node must be generated from an Unrefined Rule node. For C&R Tree models, the generated Ruleset node must be generated from the C&R Tree model node. A generated Ruleset node can be created directly by C5.0 using the ruleset option and can also be generated from a C5.0 decision tree model. Unlike Unrefined Rule nodes, generated Ruleset nodes can be placed in streams to generate predictions.

You can add the generated Ruleset node to your stream by clicking the icon in the generated models palette and then clicking the stream canvas where you want to place the node, or by right-clicking the icon and selecting Add to Stream from the context menu. Then connect your stream to the node, and you are ready to pass data to the Ruleset to generate predictions. The data coming into the generated model node must contain the same input fields, with the same types, as the training data used to create the model. (If fields are missing or field types are mismatched, you will see an error message when you execute the stream.)

To see information about the ruleset, right-click the Ruleset node and select Browse from the context menu (or Edit for nodes in a stream).

When you execute a stream containing a Ruleset node, the Ruleset node adds two new fields containing the predicted value and the confidence for each record to the data. The new field names are derived from the model name by adding prefixes. For association rulesets, the prefixes are $A-$ for the prediction field and $AC-$ for the confidence field. For C5.0 rulesets, the prefixes are $C-$ for the prediction field and $CC-$ for the confidence field. For C&R Tree rulesets, the prefixes are $R-$ for the prediction field and $RC-$ for the confidence field. In a stream with multiple Ruleset nodes in a series predicting the same output field(s), the new field names will include numbers in the prefix to distinguish them from each other. The first Association Ruleset node in the stream will use the usual names, the second node will use names starting with $A1-$ and $AC1-$, the third node will use names starting with $A2-$ and $AC2-$, and so on.
How rules are applied. Rule sets are unlike other generated model nodes because for any particular record, more than one prediction may be generated, and those predictions may not all agree. There are two methods for generating predictions from rule sets:

- **Voting.** This method attempts to combine the predictions of all of the rules that apply to the record. For each record, all rules are examined and each rule that applies to the record is used to generate a prediction and an associated confidence. The sum of confidence figures for each output value is computed, and the value with the greatest confidence sum is chosen as the final prediction. The confidence for the final prediction is the confidence sum for that value divided by the number of rules that fired for that record.

- **First hit.** This method simply tests the rules in order, and the first rule that applies to the record is the one used to generate the prediction.

The method used can be controlled in the stream options. For more information, see “Setting Options for Streams” in Chapter 4 on page 73.

Generating nodes. The generate menu allows you to create new nodes based on the ruleset.

- **Filter Node.** Creates a new Filter node to filter fields that are not used by rules in the ruleset.

- **Select Node.** Creates a new Select node to select records to which the selected rule applies. The generated node will select records for which all antecedents of the rule are true. This option requires a rule to be selected.

- **Rule Trace Node.** Creates a new SuperNode that will compute a field indicating which rule was used to make the prediction for each record. When a ruleset is evaluated using the first hit method, this is simply a symbol indicating the first rule that would fire. When the ruleset is evaluated using the voting method, this is a more complex string showing the input to the voting mechanism.

- **Single Decision Tree(Canvas)/Single Decision Tree(GM Palette).** Creates a new single Ruleset derived from the currently selected rule. Only available for **boosted C5.0 models.** For more information, see “Boosted C5.0 Models” on page 427.

- **Model to Palette.** Returns the model to the generated models palette. This is useful in situations where a colleague may have sent you a stream containing the model and not the model itself.
Chapter 12

**Generated Ruleset Model Tab**

On the generated Ruleset node Model tab, you will see a list of rules extracted from the data by the algorithm.

Figure 12-17
Sample generated Ruleset node Model tab

Rules are broken down by consequent (predicted category). When the Rule browser is opened, the rules are collapsed, so that only the rule label is visible. To unfold a category or a rule, use the expander control to the left of the item, or double-click the item. The toolbar provides controls for expanding or collapsing multiple rules.

Rules are presented in the following format:

\[
\text{if } \text{antecedent}_1 \text{ and } \text{antecedent}_2 \\
... 
\]
where consequent and antecedent_1 through antecedent_n are all conditions. The rule is interpreted as “for records where antecedent_1 through antecedent_n are all true, consequent is also likely to be true.” If you click the Show Instances/Confidence button on the toolbar, each rule will also show information on the number of records to which the rule applies—that is, for which the antecedents are true (**Instances**), and the proportion of those records for which the entire rule is true (**Confidence**).

Note that confidence is calculated somewhat differently for C5.0 rulesets. C5.0 uses the following formula for calculating the confidence of a rule:

\[
\frac{1 + \text{number of records where rule is correct}}{2 + \text{number of records for which the rule’s antecedents are true}}
\]

This calculation of the confidence estimate adjusts for the process of generalizing rules from a decision tree (which is what C5.0 does when it creates a ruleset).

**Generated Ruleset Summary Tab**

On the Summary tab of a generated Ruleset node, you will see information about the model itself (Analysis), fields used in the model (Fields), settings used when building the model (Build Settings), and model training (Training Summary).

When you first browse a generated Ruleset node, the Summary tab results start out collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or click the Expand All button to show all results. To hide results when finished viewing them, use the expander control to collapse the specific results you want to hide, or click the Collapse All button to collapse all results.
Figure 12-18
Sample generated Ruleset node Summary tab

### Analysis
The analysis section displays the tree depth. If you have executed an Analysis node attached to this modeling node, information from that analysis will also appear in this section. For more information, see “Analysis Node” in Chapter 14 on page 504.

### Fields
This section lists the fields used as the target and the inputs in building the model.

### Build Settings
This section contains information on the settings used in building the model.
**Training Summary.** This section shows the type of the model, the stream used to create it, the user who created it, when it was built, and the elapsed time for building the model.

**Generated Ruleset Settings Tab**

On the Settings tab of generated Ruleset nodes, you can specify options, such as enabling SQL generation during model scoring.

Figure 12-19
*Sample generated Ruleset node Settings tab*
**Generate SQL.** Select one of the options below to enable or disable SQL generation for the model in order to take advantage of in-database mining. The settings specified here apply only when operating with a database.

- **Do not generate.** Select to disable SQL generation for the model.
- **No missing value support.** Select to enable SQL generation without the overhead of handling missing values. This option simply sets the prediction to null ($null$) when a missing value is encountered while scoring a case. *Note:* This option is available only for decision trees and is the recommended selection for C5.0 trees or when the data has already been treated for missing values.
- **With missing value support.** Select to enable SQL generation with full missing value support. This means that SQL is generated so that missing values are handled as specified in the model. For example, C&RT trees use surrogate rules and biggest child fallback. *Note:* SQL generation does not provide efficient support for C5.0's treatment of missing values; therefore, this option is not enabled for C5.0 trees. No missing value support is recommended if you still want to generate SQL for C5.0 trees.

**Calculate Confidences.** Select to include confidences in scoring operations pushed back to the database. Control over confidences allows you to generate more efficient SQL.

**Generated Decision Tree Node**

Generated Decision Tree nodes represent the tree structures for predicting a particular output field discovered by one of the decision tree modeling nodes (C5.0, C&R Tree, or Build Rule from previous versions of Clementine). Note that although the Build Rule node has been replaced by the C&R Tree node in version 6.0 or higher, Decision Tree nodes in existing streams that were originally created using a Build Rule node will still function properly.

To see information about the decision tree model, right-click the Decision Tree node and select Browse from the context menu (or Edit for nodes in a stream).

You can add the Decision Tree node to your stream by clicking the icon in the generated models palette and then clicking the stream canvas where you want to place the node, or by right-clicking the icon and selecting Add to Stream from the context menu. Then connect your stream to the node, and you are ready to pass data to the decision tree model to generate predictions. The data coming into the generated
model node must contain the same input fields, with the same types, as the training data used to create the model. (If fields are missing or field types are mismatched, you will see an error message when you execute the stream.)

When you execute a stream containing a Decision Tree node, the Decision Tree node adds two new fields containing the predicted value and the confidence for each record to the data. The new field names are derived from the model name by adding prefixes. For C&R and Build Rule trees, the prefixes are $R-$ for the prediction field and $RC-$ for the confidence field. For C5.0 trees, the prefixes are $C-$ for the prediction field and $CC-$ for the confidence field. In a stream with multiple Decision Tree nodes in a series predicting the same output field(s), the new field names will include numbers in the prefix to distinguish them from each other. For example, the first C&R Tree node in the stream will use the usual names, the second node will use names starting with $R1-$ and $RC1-$, the third node will use names starting with $R2-$ and $RC2-$, and so on.

Generating nodes. The generate menu allows you to create new nodes based on the tree model.

- **Filter Node.** Creates a new Filter node to filter fields that are not used by the tree model. If there is a Type node upstream from this Decision Tree node, any fields with direction OUT are passed on by the generated Filter node.

- **Select Node.** Creates a new Select node to select records assigned to the currently selected branch of the tree. This option requires a tree branch to be selected.

- **Rule set.** Creates a new Ruleset node containing the tree structure as a set of rules defining the terminal branches of the tree. This option is not available when browsing a regression tree (a decision tree with a numeric output field.)

- **Single Decision Tree(Canvas)/Single Decision Tree(GM Palette).** Creates a new single Ruleset derived from the currently selected rule. Only available for boosted C5.0 models. For more information, see “Boosted C5.0 Models” on page 427.

- **Model to Palette.** Returns the model to the generated models palette. This is useful in situations where a colleague may have sent you a stream containing the model and not the model itself.

**Decision Tree Model Tab**

The Decision Tree node Model tab displays a list of conditions defining the partitioning of data discovered by the algorithm.
When the Rule browser is opened, the rules are collapsed, so that only the rule label is visible. To unfold a category or a rule, use the expander control to the left of the item, or double-click the item. The toolbar provides controls for expanding or collapsing multiple rules.

Decision trees work by recursively partitioning the data based on input field values. The data partitions are called **branches**. The initial branch (sometimes called the **root**) encompasses all data records. The root is split into subsets or **child branches**, based on the value of a particular input field. Each child branch may be further split into sub-branches, which may in turn be split again, and so on. At the lowest
level of the tree are branches that have no more splits. Such branches are known as **terminal branches**, or **leaves**.

The Decision Tree browser shows the input values that define each partition or branch and a summary of output field values for the records in that split. For splits based on numeric fields, the branch is shown by a line of the form:

```
fieldname relation value [summary]
```

where *relation* is a numeric relation. For example, a branch defined by values greater than 100 for the *revenue* field would appear as

```
revenue > 100 [summary]
```

For splits based on symbolic fields, the branch is shown by a line of the form:

```
fieldname = value [summary] or fieldname in [values] [summary]
```

where *values* are the field values that define the branch. For example, a branch that includes records where the value of *region* can be any of *North*, *West*, or *South* would be represented as

```
region in ['North' 'West' 'South'] [summary]
```

For terminal branches, a prediction is also given adding an arrow and the predicted value to the end of the rule condition. For example, a leaf defined by *revenue* > 100 that predicts a value of *high* for the output field, the Tree browser would display

```
revenue > 100 [Mode: high] high
```

The **summary** for the branch is defined differently for symbolic and numeric output fields. For trees with numeric output fields, the summary is the **average** value for the branch, and the **effect** of the branch is the difference between the average for the branch and the average of its parent branch. For trees with symbolic output fields, the summary is the **mode**, or the most frequent value, for records in the branch.

To fully describe a branch, you need to include the condition that defines the branch, plus the conditions that define the splits further up the tree. For example, in the tree

```
revenue > 100
region = "North"
region in ['South' 'East' 'West']
```
revenue <= 200

the branch represented by the second line is defined by the conditions revenue > 100 and region = "North".

If you select Show Instances/Confidence from the toolbar, each rule will also show information on the number of records to which the rule applies (Instances) and the proportion of those records for which the rule is true (Confidence).

If you select Show Additional Information Panel from the toolbar, you will see a panel containing detailed information for the selected rule at the bottom of the window. The information panel contains three tabs.

Figure 12-21
Information panel

<table>
<thead>
<tr>
<th>History</th>
<th>Value</th>
<th>Frequency</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequencies</td>
<td>drugB</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>drugX</td>
<td>6</td>
<td>55.71%</td>
</tr>
<tr>
<td></td>
<td>drugA</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>drugY</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>drugC</td>
<td>1</td>
<td>14.28%</td>
</tr>
</tbody>
</table>

- **History.** This tab traces the split conditions from the root node down to the selected node. This provides a list of conditions that determines when a record is assigned to the selected node. Records for which all the conditions are true will be assigned to this node.

- **Frequencies.** For models with symbolic target fields, this tab shows for each possible target value the number of records assigned to this node (in the training data) that have that target value. The frequency figure, expressed as a percentage (shown to a maximum of 3 decimal places) is also displayed. For models with numeric targets, this tab is empty.

- **Surrogates.** For C&R Tree models, the primary split and any surrogate splits for the selected node are shown. This tells you how records with missing values for the primary split field will be classified at that split. For other decision tree models, this tab is empty.
**Decision Tree Viewer Tab**

The Viewer tab shows a graphical display of the structure of the tree in detail. In most cases, because of the size of the overall tree, only a portion of the tree is visible in the Tree view. You can scroll the window to view other parts of the tree or use the tree map window to select a different region of the tree to view. To show the tree map window, click the tree map button on the toolbar.

**Figure 12-22**
*Sample Decision Tree Viewer tab with tree map window*

You can display each node in the tree as a table of values, a graph of values, or both. You can control the node display using the toolbar buttons. You can also change the orientation of the tree display (top-down, left-to-right, or right-to-left) using the toolbar controls.
You can expand and collapse the branches in the tree for display purposes. By default, all branches in the tree are expanded. Click the minus sign (–) next to a parent node to hide all of its child nodes. Click the plus sign (+) next to a parent node to display its child nodes.

You can select a node in the Viewer tab for generating a Ruleset or a Select node from the Generate menu. To select a node, simply click on it.

**Node statistics.** For a symbolic target field, the table shows the number and percentage of records in each category and the percentage of the entire sample that the node represents. For a range (numeric) target field, the table shows the mean, standard deviation, number of records, and predicted value of the target field.

**Node graphs.** For a symbolic target field, the graph is a bar chart of percentages in each category of the target field. Preceding each row in the table is a color swatch that corresponds to the color that represents each of the target field categories in the graphs for the node. For a range (numeric) target field, the graph shows a histogram of the target field for records in the node.

**Decision Tree Summary Tab**

On the Summary tab of a Decision Tree node, you will see information about the model itself (Analysis), fields used in the model (Fields), settings used when building the model (Build Settings), and model training (Training Summary).

When you first browse a Decision Tree node, the Summary tab results start out collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or click the Expand All button to show all results. To hide results when finished viewing them, use the expander control to collapse the specific results you want to hide, or click the Collapse All button to collapse all results.
Analysis. The analysis section displays the depth of the tree. If you have executed an Analysis node attached to this modeling node, information from that analysis will also appear in this section. For more information, see “Analysis Node” in Chapter 14 on page 504.

Fields. This section lists the fields used as the target and the inputs in building the model.

Build Settings. This section contains information on the settings used in building the model.
Training Summary. This section shows the type of the model, the stream used to create it, the user who created it, when it was built, and the elapsed time for building the model.

Generating a Ruleset from a Decision Tree

Figure 12-24
Generate Ruleset dialog box

To generate a Ruleset from a Decision Tree node, select Rule Set from the Generate menu of the Decision Tree browser. You can specify the following options for translating the tree into a ruleset:

Rule set name. Allows you to specify the name of the new generated Ruleset node.

Create node on. Controls the location of the new generated Ruleset node. Select Canvas, GM Palette, or Both.

Minimum instances. Specify the minimum number of instances (number of records to which the rule applies) to preserve in the generated ruleset. Rules with support less than the specified value will not appear in the new ruleset.

Minimum confidence. Specify the minimum confidence for rules to be preserved in the generated ruleset. Rules with confidence less than the specified value will not appear in the new ruleset.
**Boosted C5.0 Models**

**Figure 12-25**
Sample boosted C5.0 Decision Tree node Model tab

When you create a boosted C5.0 model (either a ruleset or a decision tree), you actually create a set of related models. The Model tab of the generated node browser for a boosted C5.0 model shows the list of models at the top level of the hierarchy, along with the accuracy of each model and the cumulative accuracy of the boosted models up to and including the current model. To examine the rules or splits for a particular model, select that model and expand it as you would a rule or branch in a single model.
You can also extract a particular model from the set of boosted models and create a new generated Ruleset node containing just that model. To create a new ruleset from a boosted C5.0 model, select the ruleset or tree of interest and choose either Single Decision Tree (GM Palette) or Single Decision Tree (Canvas) from the Generate menu.

**Generated Cluster Models**

Cluster models are typically used to find groups (or clusters) of similar records based on the variables examined, where the similarity between members of the same group is high and the similarity between members of different groups is low. The results can be used to identify associations that would otherwise not be apparent. For example, through cluster analysis of customer preferences, income level, and buying habits, it may be possible to identify the types of customers that are more likely to respond to a particular marketing campaign.

The following cluster models are generated in Clementine:

- Generated Kohonen net node
- Generated K-Means node
- Generated TwoStep cluster node

To see information about the generated cluster models, right-click the model node and select Browse from the context menu (or Edit for nodes in a stream).

**Cluster Viewer Tab**

The Viewer tab for cluster models shows a graphical display of summary statistics and distributions for fields between clusters.
By default, the clusters are displayed on the $x$ axis and the fields on the $y$ axis. If the cluster matrix is large, it is automatically paginated for faster display on the screen. The expanded dialog contains options for viewing all clusters and fields at once. The toolbar contains buttons used for navigating through paginated results. For more information, see “Navigating the Cluster View” on page 435.
The **cluster axis** lists each cluster in cluster number order and by default includes an **Importance** column. An **Overall** column can be added using options on the expanded dialog.

- The **Overall** column displays the values (represented by bars) for all clusters in the data set and provides a useful comparison tool. Expand the dialog using the yellow arrow button and select the **Show Overall** option.
- The **Importance** column displays the overall importance of the field to the model. It is displayed as 1 minus the $p$ value (probability value from the $t$ test or chi-square test used to measure importance).

**The field axis** lists each field (variable) used in the analysis and is sorted alphabetically. Both discrete fields and scale fields are displayed by default.

**The individual cells** of the table shows summaries of a given field's values for the records in a given cluster. These values can be displayed as small charts or as scale values.

*Note:* Some models created before Clementine 8.0 may not display full information on the Viewer tab:

- For pre-8.0 K-Means models, numeric fields always show importance as **Unknown**. Text view may not display any information for older models.
- For pre-8.0 Kohonen models, the Viewer tab is not available.

### Understanding the Cluster View

There are two approaches to interpreting the results in a cluster display:

- Examine clusters to determine characteristics unique to that cluster. *Does one cluster contain all the high-income borrowers? Does this cluster contain more records than the others?*

- Examine fields across clusters to determine how values are distributed among clusters. *Does one's level of education determine membership in a cluster? Does a high credit score distinguish between membership in one cluster or another?*

Using the main view and the various drill-down views in the Cluster display, you can gain insight to help you answer these questions.
As you read across the row for a field, take note of how the category frequency (for discrete fields) and the mean-value distribution (for range fields) varies among clusters. For example, in the image above, notice that Clusters 2 and 5 contain entirely different values for the \textit{BP} (blood pressure) field. This information, combined with the importance level indicated in the column on the right, tells you that blood pressure is an important determinant of membership in a cluster. These clusters and the \textit{BP} field are worth examining in greater detail. Using the display, you can double-click the field for a more detailed view, displaying actual values and statistics.

The following tips provide more information on interpreting the detailed view for fields and clusters.

\textbf{What Is Importance?}

For both range (numeric) and discrete fields, the higher the importance measure, the less likely the variation for a field between clusters is due to chance and more likely due to some underlying difference. In other words, fields with a higher importance level are those to explore further.

Importance is calculated as 1 minus the \textit{p} value, where probability value is taken from \textit{t} tests (for range fields) and chi-square tests (for discrete fields).
**Reading the Display for Discrete Fields**

For discrete fields, or sets, the Top View (the default cluster comparison view) displays distribution charts indicating the category counts of the field for each cluster. Drill-down (by double-clicking or using the expanded tab options) to view actual counts for each value within a cluster. These counts indicate the number of records with the given value that fall into a specific cluster.

**Figure 12-28**
*Drill-down view for a discrete field*

To view both counts and percentages, view the display as text. For more information, see “Viewing Clusters as Text” on page 441. At any time, you can click the Top View button on the toolbar to return to the main Viewer display for all fields and clusters. Use the arrow buttons to flip through recent views.
Figure 12-29
Toolbar buttons used to return to Top View and flip through recent views

Reading the Display for Scale Fields

For scale fields, the Viewer displays bars representing the mean value of a field for each cluster. The Overall column compares these mean values, but is not a histogram indicating frequency distribution. Drill-down (by double-clicking or using the expanded tab options) to view the actual mean value and standard deviation of the field for each cluster.

Figure 12-30
Drill-down view for a scale field
Chapter 12

Reading Cluster Details

You can view detailed information about a single cluster by drilling-down into the display. This is an effective way to quickly examine a cluster of interest and determine which field(s) might contribute to the cluster's uniqueness. Compare the Cluster and Overall charts by field and use the importance levels to determine fields that provide separation or commonality between clusters.

Figure 12-31
Drill-down view for a single cluster
Navigating the Cluster View

The Cluster Viewer is an interactive display. Using the mouse or the keyboard, you can:

- Drill-down to view more details for a field or cluster.
- Move through paginated results.
- Compare clusters or fields by expanding the dialog box to select items of interest.
- Alter the display using toolbar buttons.
- Scroll through views.
- Transpose axes using toolbar buttons.
- Print, copy, and zoom.
- Generate Derive, Filter, and Select nodes using the Generate button.

Using the Toolbar

You can control the display using the toolbar buttons. Move through paginated results for clusters and fields, or drill-down to view a specific cluster or field. You can also change the orientation of the display (top-down, left-to-right, or right-to-left) using the toolbar controls. You can also scroll through previous views, return to the top view, and open a dialog box to specify the colors and thresholds for displaying importance.

Figure 12-32

Toolbar for navigating and controlling the Cluster Viewer

Use your mouse on the Viewer tab to hover over a toolbar button and activate a tooltip explaining its functionality.

Moving Columns

Columns can be moved to a new position in the table by selecting one or more column headers, holding down the left mouse button, and then dragging the columns to the desired position in the table. The same approach can be taken to move rows to a new position. Note that only adjacent columns or rows can be moved together.
Generating Nodes from Cluster Models

The Generate menu allows you to create new nodes based on the cluster model. This option is available from the Model and Cluster tabs of the generated model. The generated nodes are placed unconnected on the canvas. Connect and make any desired edits before execution.

- **Filter Node.** Creates a new Filter node to filter fields that are not used by the cluster model. Make specifications by editing the Filter node generated on the stream canvas. If there is a Type node upstream from this Cluster node, any fields with direction $OUT$ are discarded by the generated Filter node.

- **Filter Node (from selection).** Creates a new Filter node to filter fields based upon selections in the Viewer. Select multiple fields using the Ctrl-click method. Fields selected in the Viewer are discarded downstream, but you may change this behavior by editing the Filter node before execution.

- **Select Node.** Creates a new Select node to select fields based upon their membership in a each cluster. A select condition is automatically generated.

- **Select Node (from selection).** Creates a new Select node to select fields based upon membership in clusters selected in the Viewer. Select multiple clusters using the Ctrl-click method.

- **Derive Node.** Creates a new Derive node, which derives a field based upon membership in all visible clusters. A derive condition is automatically generated.

- **Derive Node (from selection).** Creates a new Derive node, which derives a field based upon membership in clusters selected in the Viewer. Select multiple clusters using the Ctrl-click method.

Selecting Clusters for Display

You can specify clusters for display by selecting a cluster column in the viewer and double-clicking. Multiple adjacent cells, rows, or columns can be selected by holding down the Shift key on the keyboard while making a selection. Multiple nonadjacent cells, rows, or columns can be selected by holding down the Ctrl key while making a selection.
Alternatively, you can select clusters for display using a dialog box available from the expanded Cluster Viewer. To open the dialog box:

- Click the yellow arrow at the top of the Viewer to expand for more options.

**Figure 12-33**
*Expanded Viewer tab with Show and Sort options*

- From the Cluster drop-down list, select one of several options for display.
  - Select Display All to show all clusters in the matrix.
  - Select a cluster number to display details for only that cluster.
  - Select Clusters Larger than, to set a threshold for display clusters. This enables the Records options, which allows you to specify the minimum numbers of records in a cluster for it to be displayed.
Select Clusters Smaller than, to set a threshold for displaying clusters. This enables the Records options, which allows you to specify the maximum numbers of records in a cluster for it to be displayed.

Select Custom to hand-select clusters for display. To the right of the drop-down list, click the ellipsis (...) button to open a dialog box where you can select available clusters.

Custom Selection of Clusters

In the Show Selected Clusters dialog box, cluster names are listed in the column on the right. Individual clusters may be selected for display using the column on the left.

- Click Select All to select and view all clusters.
- Click Clear to deselect all clusters in the dialog box.

Selecting Fields for Display

You can specify fields for display by selecting a field row in the viewer and double-clicking.

Alternatively, you can select fields using a dialog available from the expanded Cluster Viewer. To open the dialog box:

- Click the yellow arrow at the top of the Viewer to expand for more options.
- From the Field drop-down list, select one of several options for display.
  - Select Display All to show all fields in the matrix.
  - Select a field name to display details for only that field.
  - Select All Ranges, to display all range (numeric) fields.
  - Select All Discrete, to display all discrete (categorical) fields.
  - Select Conditional, to display fields that meet a certain level of importance. You can specify the importance condition using the Show drop-down list.
Figure 12-34
Displaying fields based upon importance level

Select Custom to hand-select fields for display. To the right of the drop-down list, click the ellipsis (...) button to open a dialog box where you can select available fields.

**Custom Selection of Fields**

In the Show Selected Fields dialog box, field names are listed in the column on the right. Individual fields may be selected for display using the column on the left.

- Click Select All to display all fields.
- Click Clear to deselect all fields in the dialog box.

**Sorting Display Items**

When viewing cluster results as a whole or individual fields and clusters, it is often useful to sort the display table by areas of interest. Sorting options are available from the expanded Cluster Viewer. To sort clusters or fields:

- Click the yellow arrow at the top of the Viewer to expand for more options.
- In the Sort Options control box, select a sorting method. Various options may be disabled if you are viewing individual fields or clusters.
Available sort options include:

- For clusters, you can sort by size or name of the cluster.
- For fields, you can sort by field name or importance level. *Note:* Fields are sorted by importance within field type. For example, scale fields are sorted for importance first, then discrete fields.

Use the arrow buttons to specify sort direction.

**Setting Importance Options**

Using the importance dialog box, you can specify options to represent importance in the browser. Click the Importance options button on the toolbar to open the dialog box.

**Figure 12-36**
*Color options toolbar button*
Labels. To show importance labels in the cluster display, select Show labels in the Importance Settings dialog box. This activates the label text fields where you can provide suitable labels.

Thresholds. Use the arrow controls to specify the desired importance threshold associated with the icon and label.

Colors. Select a color from the drop-down list to use for the importance icon.

Icons. Select an icon from the drop-down list to use for the associated level of importance.

What Is Importance?

Importance is calculated as 1 minus the p value, where probability value is taken from t tests (for range fields) and chi-square tests (for discrete fields). This means important fields are those whose importance value is nearer to 1.

Viewing Clusters as Text

Information in the Cluster Viewer can also be displayed as text, where all values are displayed as numerical values instead of as charts.
The text view, while different in appearance, operates in the same manner as the graphical view.

To view as text:

- Click the yellow arrow at the top of the Viewer to expand for more options.
- For both Display sizes and Display distributions, you can select to view results as text.
**Generated Kohonen Node**

Generated Kohonen nodes represent the Kohonen networks created by Kohonen nodes. They contain all of the information captured by the trained network, as well as information about the Kohonen network's architecture.

To see information about the Kohonen network model, right-click the generated Kohonen node and select **Browse** from the context menu (or **Edit** for nodes in a stream).

You can add the Kohonen model to your stream by clicking the icon in the generated models palette and then clicking the stream canvas where you want to place the node, or by right-clicking the icon and selecting **Add to Stream** from the context menu. Then connect your stream to the node, and you are ready to pass data to the Kohonen model to generate predictions. The data coming into the generated model node must contain the same input fields, with the same types, as the training data used to create the model. (If fields are missing or field types are mismatched, you will see an error message when you execute the stream.)

When you execute a stream containing a generated Kohonen node, the Kohonen node adds two new fields containing the $X$ and $Y$ coordinates of the unit in the Kohonen output grid that responded most strongly to that record. The new field names are derived from the model name, prefixed by $\$KX$- and $\$KY$. For example, if your model is named *Kohonen*, the new fields would be named $\$KX-Kohonen$ and $\$KY-Kohonen$. In a stream with multiple generated Kohonen nodes in a series predicting the same output field(s), the new field names will include numbers to distinguish them from each other. The first Net node in the stream will use the usual names, the second node will use names starting with $\$KX1$- and $\$KY1$-, the third node will use names starting with $\$KX2$- and $\$KY2$-, and so on.

To get a better sense of what the Kohonen net has encoded, click the Viewer tab on the generated model browser. This displays the Cluster Viewer, providing a graphical representation of clusters, fields, and importance levels. For more information, see “Cluster Viewer Tab” on page 428.

If you prefer to visualize the clusters as a grid, you can view the result of the Kohonen net by plotting the $\$KX$- and $\$KY$- fields using a Plot node. (You should select $X$-Agitation and $Y$-Agitation in the Plot node to prevent each unit’s records from all being plotted on top of each other.) In the plot, you can also overlay a symbolic field to investigate how the Kohonen net has clustered the data.

Another powerful technique for gaining insight into the Kohonen network is to use rule induction to discover the characteristics that distinguish the clusters found by the network. For more information, see “C5.0 Node” in Chapter 11 on page 329.
**Generated Kohonen Network Model Tab**

The generated Kohonen Model tab contains detailed information about the clusters defined by the model.

**Figure 12-39**
Sample generated Kohonen node Model tab

When you first browse a generated Kohonen node, the Model tab results start out collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or use the Expand All button to show all results. To hide results when finished viewing them, use the expander control to collapse the specific results you want to hide, or use the Collapse All button to collapse all results.
Clusters. Kohonen networks, also commonly called clusters, are labeled and the number of records assigned to each cluster is shown. Each cluster is described by its center, which can be thought of as the prototype for the cluster. For scale fields, the mean value and standard deviation for training records assigned to the cluster is given; for symbolic fields, the proportion for each distinct value is reported (except for values that do not occur for any records in the cluster, which are omitted).

Generated Kohonen Network Summary Tab

On the Summary tab of a generated Kohonen node, you will see information about the network itself (Analysis), fields used in the network (Fields), settings used when building the model (Build Settings), and model training (Training Summary).

When you first browse a generated Kohonen node, the Summary tab results start out collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or use the Expand All button to show all results. To hide results when finished viewing them, use the expander control to collapse the specific results you want to hide, or use the Collapse All button to collapse all results.
Chapter 12

Figure 12-40
Sample generated Kohonen node Summary tab

Analysis. The analysis section displays information about the architecture or topology of the network. The length and width of the two-dimensional Kohonen feature map (the output layer) are shown as $KX-model_name$ and $KY-model_name$. For the input and output layers, the number of units in that layer is listed.

Fields. This section lists the fields used as inputs in building the model.

Build Settings. This section contains information on the settings used in building the model.

Training Summary. This section shows the type of the model, the stream used to create it, the user who created it, when it was built, and the elapsed time for building the model.
**Generated Models**

**Generated K-Means Node**

Generated K-Means nodes represent the clustering models created by Train K-Means nodes. They contain all of the information captured by the clustering model, as well as information about the training data and the estimation process.

To see information about the K-Means model, right-click the generated K-Means node and select **Browse** from the context menu (or **Edit** for nodes in a stream).

You can add the model to your stream by clicking the icon in the generated models palette and then clicking the stream canvas where you want to place the node, or by right-clicking the icon and selecting **Add to Stream** from the context menu. Then connect your stream to the node, and you are ready to pass data to the K-Means model to assign cluster memberships. The data coming into the generated model node must contain the same input fields, with the same types, as the training data used to create the model. (If fields are missing or field types are mismatched, you will see an error message when you execute the stream.)

When you execute a stream containing a generated K-Means node, the K-Means node adds two new fields containing the cluster membership and distance from the assigned cluster center for that record. The new field names are derived from the model name, prefixed by $KM-$ for the cluster membership and $KMD-$ for the distance from the cluster center. For example, if your model is named Kmeans, the new fields would be named $KM-Kmeans$ and $KMD-Kmeans$. In a stream with multiple generated K-Means nodes in a series predicting the same output field(s), the new field names will include numbers to distinguish them from each other. The first generated K-Means node in the stream will use the usual names, the second node will use names starting with $KM1-$ and $KMD1-$, the third node will use names starting with $KM2-$ and $KMD2-$, and so on.

A powerful technique for gaining insight into the K-Means model is to use rule induction to discover the characteristics that distinguish the clusters found by the model. For more information, see “C5.0 Node” in Chapter 11 on page 329. You can also click the Viewer tab on the generated model browser to display the Cluster Viewer, providing a graphical representation of clusters, fields, and importance levels. For more information, see “Cluster Viewer Tab” on page 428.

**Generated K-Means Model Tab**

The generated K-Means Model tab contains detailed information about the clusters defined by the model.
When you first browse a generated K-Means node, the Model tab results start out collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or use the Expand All button to show all results. To hide results when finished viewing them, use the expander control to collapse the specific results you want to hide, or use the Collapse All button to collapse all results.

Clusters. Clusters are labeled and the number of records assigned to each cluster is shown. Each cluster is described by its center, which can be thought of as the prototype for the cluster. For scale fields, the mean value for training records assigned to the cluster is given; for symbolic fields, the proportion for each distinct value is reported (except for values that do not occur for any records in the cluster,
which are omitted). If you requested Show cluster proximity in the Train K-Means node, each cluster description will also contain its proximities from every other cluster.

**Generated K-Means Summary Tab**

The generated K-Means Summary tab contains information about the training data, the estimation process, and the clusters defined by the model.

When you first browse a generated K-Means node, the Summary tab results start out collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or use the Expand All button to show all results. To hide results when finished viewing them, use the expander control to collapse the specific results you want to hide, or use the Collapse All button to collapse all results.

**Figure 12-42**
Sample generated K-Means node Summary tab
**Analysis.** The analysis section displays information about the cluster solution found by the model. The number of clusters is shown, as well as the iteration history. If you have executed an Analysis node attached to this modeling node, information from that analysis will also appear in this section. For more information, see “Analysis Node” in Chapter 14 on page 504.

**Fields.** This section lists the fields used as inputs in building the model.

**Build Settings.** This section contains information on the settings used in building the model.

**Training Summary.** This section shows the type of the model, the stream used to create it, the user who created it, when it was built, and the elapsed time for building the model.

---

**Generated TwoStep Cluster Node**

Generated TwoStep Cluster nodes represent the clustering models created by TwoStep Cluster nodes. They contain all of the information captured by the clustering model, as well as information about the training data and the estimation process.

To see information about the TwoStep cluster model, right-click the generated TwoStep node and select **Browse** from the context menu (or **Edit** for nodes in a stream). You can add the model to your stream by clicking the icon in the generated models palette and then clicking the stream canvas where you want to place the node, or by right-clicking the icon and selecting **Add to Stream** from the context menu. Then connect your stream to the node, and you are ready to pass data to the TwoStep Cluster model to assign cluster memberships. The data coming into the generated model node must contain the same input fields, with the same types, as the training data used to create the model. (If fields are missing or field types are mismatched, you will see an error message when you execute the stream.)

When you execute a stream containing a generated TwoStep Cluster node, the node adds a new field containing the cluster membership for that record. The new field name is derived from the model name, prefixed by `$T-`. For example, if your model is named `TwoStep`, the new field would be named `$T-TwoStep`. In a stream with multiple generated TwoStep Cluster nodes in series, the new field names will include numbers to distinguish them from each other. The first TwoStep node in the stream will use the usual name, the second node will use a name starting with `$T1-`, the third node will use a name starting with `$T2-`, and so on.
A powerful technique for gaining insight into the TwoStep model is to use rule induction to discover the characteristics that distinguish the clusters found by the model. For more information, see “C5.0 Node” in Chapter 11 on page 329. You can also click the Viewer tab on the generated model browser to display the Cluster Viewer, providing a graphical representation of clusters, fields, and importance levels. For more information, see “Cluster Viewer Tab” on page 428.

**Generated TwoStep Model Tab**

The generated TwoStep Model tab contains detailed information about the clusters defined by the model.

When you first browse a generated TwoStep node, the Model tab results start out collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or use the Expand All button to show all results. To hide results when finished viewing them, use the expander control to collapse the specific results you want to hide, or use the Collapse All button to collapse all results.
Clusters. Clusters are labeled, and the number of records assigned to each cluster is shown. Each cluster is described by its center, which can be thought of as the prototype for the cluster. For scale fields, the average value and standard deviation for training records assigned to the cluster is given; for symbolic fields, the proportion for each distinct value is reported (except for values that do not occur for any records in the cluster, which are omitted).

Generated TwoStep Summary Tab

The generated TwoStep Summary tab contains information about the training data, the estimation process, and the clusters defined by the model.
When you first browse a generated TwoStep node, the Summary tab results start out collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or use the Expand All button to show all results. To hide results when finished viewing them, use the expander control to collapse the specific results you want to hide, or use the Collapse All button to collapse all results.

**Figure 12-44**  
*Sample generated TwoStep node Summary tab*

### Analysis
The analysis section displays the number of clusters found.

### Fields
This section lists the fields used as inputs in building the model.

### Build Settings
This section contains information on the settings used in building the model.
Training Summary. This section shows the type of the model, the stream used to create it, the user who created it, when it was built, and the elapsed time for building the model.

Generated Sequence Rules Node

Generated Sequence Rules nodes represent the sequences found for a particular output field discovered by the Sequence node. Unlike Unrefined Rule nodes created by association rule nodes like Apriori and GRI, generated Sequence Rules nodes can be placed in streams to generate predictions.

To see information about the sequence model, right-click the Sequence Rules node and select Browse from the context menu (or Edit for nodes in a stream).

You can also add a Sequence Rules node to a data stream to generate predictions. The data format must match the format used when building the sequence model. When you execute a stream containing a Sequence Rules node, the Sequence Rules node adds a pair of fields containing predictions and associated confidence values for each prediction from the sequence model to the data. By default, three pairs of fields containing the top three predictions (and their associated confidence values) are added. You can change the number of predictions generated when you build the model by setting the Sequence node model options at build time. For more information, see “Sequence Node Model Options” in Chapter 11 on page 376.

The new field names are derived from the model name. The field names are \$S-sequence-n\ for the prediction field (where \(n\) indicates the \(n\)th prediction) and \$SC-sequence-n\ for the confidence field. In a stream with multiple Sequence Rules nodes in a series, the new field names will include numbers in the prefix to distinguish them from each other. The first Sequence Set node in the stream will use the usual names, the second node will use names starting with \$S1-\ and \$SC1-, the third node will use names starting with \$S2-\ and \$SC2-, and so on. Predictions appear in order by confidence, so that \$S-sequence-1\ contains the prediction with the highest confidence, \$S-sequence-2\ contains the prediction with the next highest confidence, and so on. For records where the number of available predictions is smaller than the number of predictions requested, remaining predictions contain the value \$null\$. For example, if only two predictions can be made for a particular record, the values of \$S-sequence-3\ and \$SC-sequence-3\ will be \$null\$. 
Predictions made by the generated Sequence Model node are not tied to the time stamp of the record to which they are added. They simply refer to the most likely items to appear at some point in the future, given the history of transactions for the current ID up to the current record. For more information, see “Predictions from Sequence Rules” on page 455.

Note: When scoring data using a generated Sequence Set node in a stream, any tolerance or gap settings that you selected in building the model are ignored for scoring purposes.

Generating nodes. The Generate menu allows you to create new SuperNodes based on the sequence model.

- Rule SuperNode. Generates a SuperNode that can detect and count occurrences of sequences in scored data. This option is disabled if no rule is selected. For more information, see “Generating a Rule SuperNode from a Sequence Rules Node” on page 460.

- Model to Palette. Returns the model to the generated models palette. This is useful in situations where a colleague may have sent you a stream containing the model and not the model itself.

Predictions from Sequence Rules

When you pass data records into a Sequence Rules node, the node handles the records in a time-dependent manner (or order-dependent, if no timestamp field was used to build the model). Records should be sorted by the ID field and timestamp field (if present).

For each record, the rules in the model are compared to the set of transactions processed for the current ID so far, including the current record and any previous records with the same ID and earlier timestamp. The $k$ rules with the highest confidence values that apply to this set of transactions are used to generate the $k$ predictions for the record, where $k$ is the number of predictions specified when the model was built. (If multiple rules predict the same outcome for the transaction set, only the rule with the highest confidence is used.)

Note that the predictions for each record do not necessarily depend on that record's transactions. If the current record's transactions do not trigger a specific rule, rules will be selected based on the previous transactions for the current ID. In other words, if the current record doesn't add any useful predictive information to the sequence,
the prediction from the last useful transaction for this ID is carried forward to the current record.

For example, suppose you have a Sequence Rule model with the single rule

Jam -> Bread (0.66)

and you pass it the following records:

<table>
<thead>
<tr>
<th>ID</th>
<th>Purchase</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>jam</td>
<td>bread</td>
</tr>
<tr>
<td>001</td>
<td>milk</td>
<td>bread</td>
</tr>
</tbody>
</table>

Notice that the first record generates a prediction of bread, as you would expect. The second record also contains a prediction of bread, because there's no rule for jam followed by milk; therefore the milk transaction doesn't add any useful information, and the rule Jam -> Bread still applies.

**Sequence Rules Model Tab**

On the Model tab of a Sequence Rules node, you will see a table containing the rules extracted by the algorithm. Each row in the table represents a rule. The first column represents the consequent (the “then” part of the rule), and subsequent columns represent the antecedents (the “if” part of the rule). Each antecedent represents one itemset in the sequence, and the itemsets are shown in the order in which they appear in the sequence.
Each rule is shown in the following format:

**Consequent**  
 frozenmeal  
 cannedveg

**Antecedent 1**  
 frozenmeal  
 frozenmeal

**Antecedent 2**  
 beer

The first example rule is interpreted as **for IDs that had “frozenmeal” in one transaction and then “beer” in another, there is likely a subsequent occurrence of “frozenmeal.”** The second example rule is interpreted as **for IDs that had “frozenmeal” and “beer” in the same transaction, there is likely a subsequent occurrence of “cannedveg.”** There is an important difference between the two rules: in the first rule, frozenmeal and beer are purchased at different times, but in the second rule they are purchased at the same time.
If you select Show Instances/Confidence from the toolbar, each rule will also show information on the number of IDs for which the sequence appears (Instances), the proportion of the training data IDs represented by the instances (Support), and the proportion of those IDs for which the rule is true (Confidence). Note that the proportions are based on valid transactions (transactions with at least one observed item or true value) rather than total transactions. Invalid transactions—those with no items or true values—are discarded for these calculations.

**Sort menu.** The Sort menu controls sorting of the rules. Direction of sorting (ascending or descending) can be changed using the sort direction button on the toolbar. Select from the following sort keys:

- **Support.** Sorts rules by support.
- **Confidence.** Sorts rules by confidence. This is the default.
- **Support * Confidence.** Sorts rules by the product of support and confidence. This emphasizes rules that are both accurate and apply to a large proportion of the training data.
- **Consequent.** Sorts rules alphabetically by the predicted value (the consequent).
- **First Antecedent.** Sorts rules by the first item of the first antecedent itemset.
- **Last Antecedent.** Sorts rules by the first item of the last antecedent itemset.
- **Number of Items.** Sorts rules by the number of items, counting individual items across all itemsets in the rule.

For example, the following table is sorted in ascending order by number of items:

<table>
<thead>
<tr>
<th>Consequent</th>
<th>Antecedent 1</th>
<th>Antecedent 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>cannedveg</td>
<td>cannedveg</td>
<td>frozenmeal</td>
</tr>
<tr>
<td>frozenmeal</td>
<td>cannedveg and frozenmeal</td>
<td>and beer</td>
</tr>
</tbody>
</table>

The first rule has two antecedents (itemsets) and two items. The second rule has only one antecedent itemset, but that itemset contains three items, so this rule comes after the first rule.
Sequence Set Summary Tab

On the Summary tab of a Sequence Rules node, you will see information about the model itself (Analysis), fields used in the model (Fields), settings used when building the model (Build Settings), and model training (Training Summary).

When you first browse a Sequence Rules node, the Summary tab results start out collapsed. To see the results of interest, use the expander control to the left of the item to show the results, or click the Expand All button to show all results. To hide results when finished viewing them, use the expander control to collapse the specific results you want to hide, or click the Collapse All button to collapse all results.

Figure 12-46
Sample Sequence Rules node Summary tab

Analysis. The analysis section displays the number of rules discovered and the minimum and maximum for support and confidence of rules in the rules. If you have executed an Analysis node attached to this modeling node, information from that analysis will also appear in this section. For more information, see “Analysis Node” in Chapter 14 on page 504.
Fields. This section lists fields used as ID field, Time field (if any), and Content field(s).

Build Settings. This section contains information on the settings used in building the model.

Training Summary. This section shows the type of the model, the stream used to create it, the user who created it, when it was built, and the elapsed time for building the model.

Generating a Rule SuperNode from a Sequence Rules Node

To generate a Rule SuperNode from a Sequence Rules node, select Rule SuperNode from the Generate menu of the Sequence Rules browser.

Important: To use the generated SuperNode, you must sort the data by ID field (and Time field, if any) before passing them into the SuperNode. The SuperNode will not detect sequences properly in unsorted data.

You can specify the following options for generating a Rule SuperNode:

Detect. Specifies how matches are defined for data passed into the SuperNode.
**Antecedents only.** The SuperNode will identify a match any time it finds the antecedents for the selected rule in the correct order within a set of records having the same ID, regardless of whether the consequent is also found. Note that this does not take into account timestamp tolerance or item gap constraint settings from the original Sequence modeling node. When the last antecedent itemset is detected in the stream (and all other antecedents have been found in the proper order), all subsequent records with the current ID will contain the summary selected below.

**Entire sequence.** The SuperNode will identify a match any time it finds the antecedents and the consequent for the selected rule in the correct order within a set of records having the same ID. This does not take into account timestamp tolerance or item gap constraint settings from the original Sequence modeling node. When the consequent is detected in the stream (and all antecedents have also been found in the correct order), the current record and all subsequent records with the current ID will contain the summary selected below.

**Display.** Controls how match summaries are added to the data in the Rule SuperNode output.

- **Consequent value for first occurrence.** The value added to the data is the consequent value predicted based on the first occurrence of the match. Values are added as a new field named `rule_n_consequent`, where `n` is the rule number (based on the order of creation of Rule SuperNodes in the stream).

- **True value for first occurrence.** The value added to the data is true if there is at least one match for the ID and false if there is no match. Values are added as a new field named `rule_n_flag`.

- **Count of occurrences.** The value added to the data is the number of matches for the ID. Values are added as a new field named `rule_n_count`.

- **Rule number.** The value added is the rule number for the selected rule. **Rule numbers** are assigned based on the order in which the SuperNode was added to the stream. For example, the first Rule SuperNode is considered `rule 1`, the second Rule SuperNode is considered `rule 2`, etc. This option is most useful when you will be including multiple Rule SuperNodes in your stream. Values are added as a new field named `rule_n_number`.

- **Include confidence figures.** If selected, this option will add the rule confidence to the data stream as well as the other summary selected above. Values are added as a new field named `rule_n_confidence`. 
Exporting Overview

There are several ways to export models built in Clementine. This section discusses two methods:
- Exporting the model and stream together
- Exporting only the model as C code or PMML

Exporting Models and Streams

Many data mining applications offer support for exporting models, but few offer support for the complete deployment of data preparation, manipulation, and modeling. Clementine provides several ways for you to export the entire data mining process to an application such as PredictiveMarketing or your own application using Clementine Solution Publisher. Essentially, the work you do in Clementine to prepare data and build models can be used to your advantage outside of Clementine.

You can export an entire stream and model in the following ways:
- Use a Publisher node to export the stream and model for later use with the Clementine Solution Publisher Runtime. For more information, see “Solution Publisher Node” in Chapter 14 on page 531.
- Use a wizard in Clementine to package the stream components for export to the PredictiveMarketing application.
Chapter 13

Exporting Models

Most models generated in Clementine can be exported individually in either C code or as XML model files encoded as predictive model markup language (PMML).

Table 13-1
Models and available types of export

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Supported Export Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Net</td>
<td>C code, PMML</td>
</tr>
<tr>
<td>Build C5.0</td>
<td>C code, PMML</td>
</tr>
<tr>
<td>Kohonen</td>
<td>C code, PMML</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>PMML</td>
</tr>
<tr>
<td>Generalized Rule Induction (GRI)</td>
<td>PMML</td>
</tr>
<tr>
<td>Apriori</td>
<td>PMML</td>
</tr>
<tr>
<td>K-Means</td>
<td>C code, PMML</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>PMML</td>
</tr>
<tr>
<td>TwoStep Cluster</td>
<td>PMML</td>
</tr>
<tr>
<td>Classification and Regression (C&amp;R) Trees</td>
<td>PMML</td>
</tr>
<tr>
<td>Sequence Detection</td>
<td>PMML</td>
</tr>
<tr>
<td>Factor Analysis/PCA</td>
<td>none</td>
</tr>
</tbody>
</table>

To export a model:

Once you have generated a model, you can export it as C code or encoded as PMML using the following steps:

► Right-click a model on the Models tab in the managers window.

► From the context menu, choose Export PMML or Export C code.
PMML Export

All models in Clementine, except Factor/PCA models, can be exported as PMML. Exporting a model using the Export PMML option produces an XML file containing a PMML 2.1 description of the model.

For PMML model export, you should choose the name of the XML file to be created or overwritten.

For more details about PMML, see the data mining group Web site at http://www.dmg.org.

C Code Export

Four models in Clementine can be exported as C code—Neural Net, Kohonen, C5.0, and K-Means. When exporting as C code, choose the name of the C source file, such as mysource.c. A header file is also exported with the same name but with an .h extension (for example, mysource.h). Depending on the model type, a data file named mysource.san or mysource.sak may also be exported.

Note: Because the code includes field names and data values, exported C code for models built using some languages, particularly those that involve double-byte character sets, may require additional modification before compiling.
Chapter 13

Exporting C5.0 Decision Trees and Rulesets

Rule export creates two files: a header file and a source file. The names of these files correspond to the name of the OUT field predicted by the rule. For example, if the name of the OUT field is NAME, exporting generates the files NAME.h and NAME.c.

Rule Header File

The header file defines constants to represent symbolic (set and flag) field values. Names for these constants have the following form:

FIELD_VALUE

For example, if field BP has a value of high, this will be represented as BP_high.

In addition, the header file defines a struct to represent an example. The name of this struct is exNAME, where NAME corresponds to the OUT field. The struct handles integer fields as int and real fields as float.

Rule Source File

The source file contains C code used to make predictions for the associated model. This code begins by including the corresponding header file. The remainder of the source file defines the function for prediction. The name of this function corresponds to the OUT field. The example structure defined in the header file serves as an argument to the function.

For example, if the OUT field is NAME, the function has the following structure:

int NAME(struct exNAME example, double *confidence) {
  ....
}

For decision trees, the function body { .... } consists of “cascaded” IF and SWITCH statements. Rule sets translate to single-level conditionals. Boosted models generate one “static” (file local) function per model.

The value returned by the function depends on the OUT field type:

- A real OUT field yields a float.
- An integer OUT field returns an int.
A symbolic OUT field yields an integer corresponding to one of the constants defined for the field’s values in the header file.

An “unclassifiable” example returns a 0 (defined by the constant unknown). The function automatically sets the confidence to the value associated with the classification.

Exporting Nets

Net export creates three files:

- Wrapper header file
- File of wrapper routines
- Network definition file (*.san)

The wrapper routines utilize functions found in other files that do not depend on your model. These functions use the network definition file to reproduce the neural network.

The names of the files depend on the name of the neural network. For example, for a network named DRUG, exporting creates files named DRUG.h, DRUG.c, and DRUG.san.

Neural Net Wrapper Header File

The exported neural net wrapper header file declares two structs and two functions, with names derived from the network name. For example, for a network named Drug, the header file contains:

- Drug_inputs, a struct for inputs
- Drug_outputs, a struct for outputs
- Two functions for handling input and output

The first function:

```c
void neural_Drug_CLEARSETS ()
```
clears the input sets data. The other function:

```c
struct Drug_outputs *neural_Drug( struct Drug_inputs *ins,
                        struct Drug_outputs *outs )
```

takes an input structure and returns the results from the network in an output structure.

**Neural Net Wrapper Routines**

The source file contains the code for the two functions defined in the header file. This code contains calls to functions found in `neurals.c`.

The value returned by the wrapper routines depends on the `OUT` field:

- A set returns a *char.*
- A flag yields an int (1 for true, 0 for false).
- A real returns a double.
- An integer yields an int.

**Other Files for Neural Net Export**

In order to use a network, additional files must be included during compilation. These files are found in the `LIB/sann` subdirectory of your Clementine installation and are named `forprop.h`, `forprop.c`, `neurals.h`, and `neurals.c`.

The file `neurals.c` provides five functions for network processing:

- int neural_init(char *filename)

  Opens a network whose definition corresponds to the named file (normally a *.san file).

- int neural_set(char *netname, char*varnam, ...)

  Sets the variable `varnam` in the network `netname` to the third argument, whose type depends on the variable being set.

- int neural_propagate(char *netname)
Forward-propagates the values on the inputs through the network `netname` to the output.

- int neural_result(char *netname, char *varname, struct neural_r_u *results)

Puts the result from the output variable `varname` in the network `netname` into the union `results` (`results.c` for sets, `results.d` for real ranges, or `results.i` for integers and flags). See `struct neural_r_u` in the `neurals.h` header file.

- int neural_close(char *netname)

Closes the network named `netname`.

**Neural Net Export Example**

The following is an example of the code that should be written to get results from the network. This file is called, arbitrarily, `find_drug.c`.

```c
#include "neurals.h"
#include "Drug.h"

int main( int argc, char **argv )
{
    struct Drug_inputs ins;
    struct Drug_outputs outs;

    neural_init("Drug.san");
    ins.Age = 39;
    ins.Sex = 1;
    ins.BP = "NORMAL";
    ins.Cholesterol = 0;
    ins.Na_to_K = 13.1457;

    neural_Drug(&ins, &outs);

    printf("\nResult is %s with confidence %f\n\n", 
            outs.Drug, outs.Drug_confidence);

    neural_close("Drug");
    return 0;
}
```

In the above example, flag fields (symbolic fields with a maximum of two possible values) equal 1 or 0 (for true and false, respectively); use a Type node in the stream from which the code was generated to ascertain these values.
This example can be compiled (with an ANSI C compiler) with a command such as:

```bash
c make find_drug .c Drug .c neurals .c forprop .c -lm
```

Consult your C manual for compiler-specific details.

Without using the wrapper routine, the code would be as follows. The file is called, arbitrarily, `find_drug_no_wrapper .c`.

```c
#include "neurals.h"

int main(int argc, char **argv )
{
    struct neural_r_u res;
    neural_init("Drug .san");
    neural_set("Drug", "BP", "CLEARSET");
    neural_set("Drug", "Age", 39);
    neural_set("Drug", "Sex", 1);
    neural_set("Drug", "BP", "NORMAL");
    neural_set("Drug", "Cholestrol", 0);
    neural_set("Drug", "Na_to_K", 13.1457);
    neural_propagate("Drug");
    neural_result("Drug", "Drug", &res);
    printf("\nResult is %s with confidence %f\n\n", res .c, res .confidence);

    neural_close("Drug");
}
```

To compile this (with an ANSI C compiler), the command would be something like:

```bash
c make find_drug_no_wrapper .c neurals .c forprop .c -lm
```

**Exporting Radial Basis Function (RBF) Nets**

As with other neural networks, RBF export creates three files:

- Wrapper header file
- File of wrapper routines
- Network definition file (*.san)
The wrapper routines utilize functions found in `neurals_rbf.c`. These functions use the network definition file to reproduce the network.

RBF net export is based on the name of the RBF network. For example, for an RBF network named Drug, three files are written: Drug.h, Drug.c, and Drug.san.

**RBF Net Wrapper Header File**

The exported RBF net wrapper header file declares two structs and two functions, with names derived from the network name. For example, a network named Drug yields a header file containing:

- Drug_inputs, a struct for inputs
- Drug_outputs, a struct for outputs
- Two functions for handling input and output

The first function:

```c
void neural_Drug_CLEARSETS ()
```

clears the input sets data. The second function:

```c
struct Drug_outputs *neural_Drug( struct Drug_inputs *ins, 
struct Drug_outputs *outs )
```

takes an input structure and returns the results from the RBF network in an output structure.

**RBF Net Wrapper Routines**

The source file contains the code for the two functions defined in the corresponding header file. This code contains calls to functions found in `neurals_rbf.c`.

The value returned by the wrapper routines depends on the OUT field:

- A set yields `*char`.
- A flag returns an int (1 for true, 0 for false).
- A real yields a double.
- An integer returns an int.

**Other Files for RBF Net Export**

In order to use a network, you must include additional files during compilation. These files are found in the LIB/sann subdirectory of your Clementine installation and are named rbfprop.h, rbfprop.c, neurals_rbf.h, and neurals_rbf.c.

The file neurals_rbf.c provides five functions for network processing:

- int neural_rbf_init(char *filename)
  
  Opens an RBF network defined in the named file (normally a *.san file).

- int neural_rbf_set(char *netnam, char *varnam, ...)
  
  Sets the variable varnam in the network netnam to the third argument, whose type depends on the variable being set.

- int neural_rbf_propagate(char *netname)
  
  Forward-propagates the values on the inputs through the network netname to the output.

- int neural_rbf_result(char *netname, char *varname, struct neural_r_u *results)
  
  Puts the result from the output variable varname in the network netname into the union results (results.c for sets, results.d for real ranges, or results.i for integers and flags). See struct neural_r_u in neurals_rbf.h.

- int neural_rbf_close(char *netname)
  
  Closes the network named netname.

**Exported RBF Net Example**

The following is an example of the code that should be written to get results from the network. This file is called, arbitrarily, find_drug.c.

```c
#include "neurals_rbf.h"
#include "Drug.h"
```
int main( int argc, char **argv )
{
    struct Drug_inputs ins;
    struct Drug_outputs outs;

    neural_rbf_init("Drug.san");

    ins.Age = 39;
    ins.Sex = 1;
    ins.BP = "NORMAL";
    ins.Cholesterol = 0;
    ins.Na_to_K = 13.1457;

    neural_Drug(&ins, &outs);

    printf("\nResult is %s with confidence %f\n\n",
           outs.Drug, outs.Drug_confidence);

    neural_rbf_close("Drug");
    return 0;
}

In the above example, flag fields (symbolic fields with a maximum of two possible values) equal 1 or 0 for true and false, respectively; use a Type node in the stream from which the code was generated to determine these values.

This example can be compiled (with an ANSI C compiler) with a command such as:

cc find_drug.c Drug.c neurals_rbf.c rbfprop.c -lm

Consult your C manual for compiler-specific details.

Without using the wrapper routine, the code would be as follows. The file is called, arbitrarily, find_drug_no_wrapper.c.

#include "neurals_rbf.h"

int main(int argc, char **argv )
{
    struct neural_r_u res;

    neural_rbf_init("Drug.san");
    neural_rbf_set("Drug","BP","CLEARSET");
    neural_rbf_set("Drug","Age",39);
    neural_rbf_set("Drug","Sex",1);
    neural_rbf_set("Drug","BP","NORMAL");
    neural_rbf_set("Drug","Cholestrol",0);
    neural_rbf_set("Drug","Na_to_K",13.1457);

neural_rbf_propagate("Drug");
neural_rbf_result("Drug","Drug",&res);

printf("\nResult is %s with confidence %f\n\n", res.c, res.confidence);

neural_rbf_close("Drug");
return 0;
}

To compile this (with an ANSI C compiler), the command would be something like:

cc find_drug_no_wrapper.c neurals_rbf.c rbfprop.c -lm

Exporting Kohonen Nets

Kohonen net export creates three files:

- Wrapper header file
- File of wrapper routines
- Network definition file (*.san)

The wrapper routines utilize functions found in koh_net.c. These functions use the network definition file to reproduce the network.

Kohonen net export is based on the name of the Kohonen network. For example, for a network named Cluster_Drug, three files are written: Cluster_Drug.h, Cluster_Drug.c, and Cluster_Drug.san.

Kohonen Net Wrapper Header File

The exported Kohonen net wrapper header file declares two structs and two functions, with names derived from the network name. For a network named Cluster_Drug, the header file contains:

- Cluster_Drug_inputs, a struct for inputs
- Cluster_Drug_outputs, a struct for outputs from the topology map
- Two functions for handling input and output
The first function:

```c
void kohonen_Cluster_Drug_CLEARSETS ()
```

clears the input sets data. The second function:

```c
struct Cluster_Drug_outputs *kohonen_Cluster_Drug( struct Cluster_Drug_inputs *ins, struct Cluster_Drug_outputs *outs )
```
takes an input structure and returns the results from the network in an output structure.

**Kohonen Net Wrapper Routines**

The source file contains the code for the functions defined in the header file. The code contains calls to functions found in `koh_net.c`.

The output struct contains integer fields called `dimension1` to `dimensionN`, where `N` is the dimension in the output map.

**Other Files for Kohonen Net Export**

In order to use a network, you must include other files during compilation. These files are found in the `LIB/sann` subdirectory of your Clementine installation and are named `kohprop.h`, `kohprop.c`, `koh_net.h`, and `koh_net.c`.

The file `koh_net.c` provides five functions:

- `int kohonen_init(char *filename)`
  Opens the Kohonen network defined in the named file (normally a `*.san` file).

- `int kohonen_set(char *netname, char *varnam, ...)`
  Sets the variable `varnam` in the network `netname` to the third argument, whose type depends on the variable being set.

- `int kohonen_propagate(char *netname)`
  Forward-propagates the values on the inputs through the network `netname` to the output.

- `int kohonen_result( char *netname, char *varname, int *results )`
Chapter 13

Puts the result from the output variable `varname` in the network `netname` into the integer array `results`.

```c
int kohonen_close(char *netname)
```

Closes the network `netname`.

**Exported Kohonen Net Example**

The following is an example of the code that should be written to get results from the network. This file is called, arbitrarily, `find_drug.c`.

```c
#include "koh_net.h"
#include "Cluster_Drug.h"

int main( int argc, char **argv )
{
    struct Cluster_Drug_inputs ins;
    struct Cluster_Drug_outputs outs;

    kohonen_init("Cluster_Drug.san");
    ins.Age = 39;
    ins.Sex = 1;
    ins.BP = "Normal";
    ins.Cholesterol = 0;
    ins.Na_to_K = 13.1457;

    kohonen_Cluster_Drug(&ins, &outs);

    printf("\nMap co-ords : %i %i\n", outs.dimension1, outs.dimension2);

    kohonen_close("Cluster_Drug");
    return 0;
}
```

In this example, flag fields (symbolic fields with a maximum of two possible values) equal 1 or 0 for true and false, respectively; use a Type node in the stream from which the code was generated to determine these values.

This code can be compiled (with an ANSI C compiler) with a command such as:

```bash
c g find_drug.c Cluster_Drug.c koh_net.c kohprop.c -lm
```

Consult your C manual for compiler-specific details.
Exporting Models

Without using the wrapper routine, the code would be as follows. The file is called, arbitrarily, `find_drug_no_wrapper.c`.

```c
#include "koh_net.h"

int main( int argc, char **argv )
{
    int *res = (int*)calloc(2,sizeof(int));
    kohonen_init("Cluster_Drug.san");
    kohonen_set("Cluster_Drug","BP","CLEARSET");
    kohonen_set("Cluster_Drug","Age",39);
    kohonen_set("Cluster_Drug","Sex",1);
    kohonen_set("Cluster_Drug","BP","NORMAL");
    kohonen_set("Cluster_Drug","Cholesterol",0);
    kohonen_set("Cluster_Drug","Na_to_K",13.1457);

    kohonen_propagate("Cluster_Drug");
    kohonen_result("Cluster_Drug","Drug",&res);
    printf("\nMap co-ords : %i %i\n", res[0], res[1]);

    kohonen_close("Cluster_Drug");
    free(res);
    return 0;
}
```

To compile this (with an ANSI C compiler), the command would be something like:

```
cc find_drug_no_wrapper.c koh_net.c kohprop.c -lm
```

Exporting K-Means Models

The export of K-Means models creates three files:
- Wrapper header file
- File of wrapper routines
- Centers definition file (*.sak)

The wrapper routines utilize functions found in `genkm.c`. A function in `genkm.c` automatically reads the *.sak file, which contains the following information (for N clusters of M inputs):
- Number of clusters
<table>
<thead>
<tr>
<th>Number of inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 1 coordinates for inputs 1 through M</td>
</tr>
<tr>
<td>Cluster 2 coordinates for inputs 1 through M</td>
</tr>
<tr>
<td>Cluster N coordinates for inputs 1 through M</td>
</tr>
</tbody>
</table>

K-Means model export is based on the name of the model. For example, for a model named Drug, three files are written: KM_Drug.h, KM_Drug.c, and KM_Drug.sak.

**K-Means Model Wrapper Header File**

The header file defines constants to represent symbolic (set and flag) field values. Names for these constants have the following form:

`FIELD_VALUE`

For example, if field `BP` has the value `high`, this will be represented as `BP_high`.

In addition, the header file defines a struct to represent an example. The name of this struct is `exNAME`, where `NAME` corresponds to the name of the model. The struct handles integer fields as int and real fields as double.

**K-Means Model Wrapper Routines**

This file contains the code for two functions, which contain calls to the functions found in `genkm.c`. The first function:

```c
void encode_Drug ( struct kmDrug example, double *ex )
```

encodes an example using values between 0 and 1, and puts the result in a double array `ex`. The second function:

```c
int calc_Drug( struct kmDrug example, struct Kmeans km, double *distance)
```

uses an example and an array of centers (contained in the struct `Kmeans`) to compute the distance between the example and each center, returning the minimum distance found and the corresponding cluster number.
**Other Files for K-Means Model Export**

In order to use an exported model, you must include the files `kmgen.h` and `kmgen.c` in the compilation. The file `kmgen.h` provides the definition of the struct `Kmeans`, containing the slots:

- `nb_centers`, the number of clusters
- `nb_inputs`, the number of inputs to the model
- `centers`, the clusters' centers values

The other file, `kmgen.c`, provides two functions. The first:

```c
void km_init( char* name, struct Kmeans *km )
```

opens the file `name` (the `.sak` file) and fills the struct `Kmeans`. The second function:

```c
void km_close(struct Kmeans *km)
```

closes the model.

**Exported K-Means Model Example**

The following is an example of the code that should be written to get results from the network. This file is called, arbitrarily, `find_drug.c`.

```c
#include "genkm.h"
#include "KM_Drug.h"

int main( int argc, char **argv )
{
    struct kmDrug example;
    struct Kmeans km;
    double distance;
    int i,clu;

    km_init("KM_Drug.sak", &km);

    example.Age = 60;
    example.Sex = Sex_M;
    example.BP = BP_NORMAL;
    example.Cholesterol = Cholesterol_NORMAL;
    example.Na_to_K = 10.09;
    example.Drug = Drug_drugX;
    clu = calc_Drug(example, km, &distance);
```
printf("Cluster_%d, distance = %lf\n",clu,distance);

km_close(&km);
return 0;
}

This can be compiled (with an ANSI C compiler) with a command such as:

cc genkm.c KM_Drug.c find_drug.c -lm

Consult your C manual for compiler-specific details.

**Field Names**

Whenever possible, the code contained in exported models uses the field names found in the model. However, in order to produce valid C code, some names may be modified as follows:

- Names of set fields that begin with a number, such as 12_inch, receive an underscore prefix, _12_inch.
- Non-alphanumeric characters in field names become underscore characters in the exported code. For example, Barcelona->Diagonal becomes Barcelona__Diagonal.

Notice that these modifications may result in undesired redundancies in the field names. For example, the two fields _1_myfield and -1_myfield both receive new names of _1_myfield. To avoid these naming conflicts, use field names that begin with a letter and contain alphanumeric characters only, or use a Filter node to assign a new name to any field that would otherwise be changed.

**Error Codes for Model Export**

Exported code for neural nets, RBF nets, and Kohonen nets use functions external to the particular network being exported. If difficulties arise during subsequent compilation or execution, these functions return the following error codes:

- 0: OK
- 1: FILE NOT FOUND
Exporting Models

- 2: INVALID NETWORK NAME
- 3: NOT ENOUGH AVAILABLE MEMORY
- 4: NO SUCH VARIABLE IN THE NETWORK
- 5: WRONG DIRECTION FOR VARIABLE
- 6: INPUT VARIABLE ASSIGNMENT INCORRECT
- 7: UNKNOWN GENERIC TYPE (PROBABLY A FILE FAULT)
- 8: NETWORK ALREADY IN MEMORY
- 9: NO MORE RESULTS AVAILABLE FROM AN OUTPUT SET

Exporting to PredictiveMarketing

Using options within Clementine, you can easily publish streams for use with the PredictiveMarketing system, a Web-enabled predictive analytics solution from SPSS. All of the data manipulation and modeling work done in Clementine can quickly be packaged as a scoring solution and saved directly into the PredictiveMarketing product.

To help you package streams, the PredictiveMarketing Wizard has been added to Clementine. To access the Wizard, from the menus choose:

Tools
PredictiveMarketing

This opens the PredictiveMarketing Wizard, which walks you through the steps of specifying fields and other import information needed by PredictiveMarketing.

PredictiveMarketing Wizard Overview

Before getting started with the PredictiveMarketing Wizard, there are a couple of concepts you should be familiar with. The following terms are used when referring to the modeling stream and published modeling scenario.
**What is a scenario?**

A **scenario** is a collection of tools that provides you with the analytical capabilities to address a particular business issue. For example, suppose you want to determine which customers are likely to respond to a summer sale campaign. You can use a scenario to obtain data from your database, create a predictive model, score the model, view reports, and explore the model.

A scenario encompasses the following components:

**Dataset.** The dataset contains metadata information that is the bridge between the data in your database and the information required to build the model.

**Model.** The model is the heart of the scenario. The model processes your data using a specific algorithm and creates a framework for analysis of new data. This is the scenario component built using Clementine.

**Reports.** Reports viewed in PredictiveMarketing provide an analysis of the model results. However, reports are not available for imported Clementine and imported PMML models.

**Exploration.** The Model Explorer allows you to differentiate customers by modifying the criteria for a customer profile. For C&RT models, the system returns a score for the customer profile. For Association Rules models, the system describes which rules apply to the customer profile.

**What is a scenario template?**

The framework for a PredictiveMarketing scenario, called the **scenario template**, is what you are creating using the PredictiveMarketing Wizard in Clementine. The scenario template determines the basic characteristics of a scenario. For example, the scenario template dictates the data set, model type, and the fields that appear on the PredictiveMarketing Scenario Explorer page.

All of these components are packaged as a .jar file using the Wizard. Now you need to describe the application of the target template as well as a couple of parameters.

After the scenario template is generated from Clementine, it is published to PredictiveMarketing, where various parameters can be edited with a tool called the **Scenario Explorer**. Here users create new scenarios, choosing from any of the available scenario templates that have been deployed. The new scenario uses
the scenario template as a foundation rather than modifying the scenario template itself. To modify the template itself, you need to use Clementine to republish the stream and model.

**Step 1: Welcome Screen**

When you first open the PredictiveMarketing Wizard, a welcome screen appears, orienting you to the process of bundling the necessary stream components.

*Figure 13-2
Welcome to the PredictiveMarketing Wizard*

The rest of the PredictiveMarketing Wizard pages walk you through the process of generating a template for deployment into the PredictiveMarketing application. Help is provided on each screen, orienting you to the given task. For additional hints on each screen, click the Help button to open the relevant topic in the online Help.
Chapter 13

Exporting to Cleo

Using options within Clementine, you can easily publish streams for use with Cleo, a customizable solution that allows you to extend the power of predictive analytics to a Web-based audience. The Cleo interface can be completely customized for your target application and operates seamlessly within the SPSS Web Deployment Framework.

To help you package streams, the Cleo Wizard has been added to Clementine. To access the Wizard, from the menus choose:

Tools
  Cleo

This opens the Cleo Wizard, which walks you through the steps of specifying fields and other import information needed by Cleo.

Cleo Wizard Overview

Before getting started with the Cleo Wizard, there are a couple of concepts you should be familiar with. The following terms are used when referring to the modeling stream and published modeling scenario.

What is a Cleo scenario?

A Cleo scenario is a collection of tools that provides you with the analytical capabilities to address a particular business issue, such as real-time churn prediction or credit scoring. You can use a Cleo scenario to obtain data from your database or to enter a single record for analysis. Then, you can score the data on real-time, using a model created in Clementine—all without purchasing or distributing Clementine to the target user.

The interface for a Cleo scenario is entirely customizeable, depending on the settings you specify in the Cleo Wizard and the options described in the Cleo Implementation Guide, available with the Cleo product.

What is a scenario bundle?

The framework for a scenario, called the scenario bundle, is what you are creating using the Cleo Wizard in Clementine. The scenario bundle determines the components of a scenario as well as an XML definition of its Web interface. For
example, the scenario bundle dictates requirements such as the type of data input and any required fields. It also contains a blueprint for the look of the Cleo Web pages, specifying items such as whether the input Web page will contain a drop-down list of field values or a radio button control.

All of these components are packaged as a .jar file using the Wizard. Now you need to provide metadata describing how the stream and fields will be used as well as a couple of formatting options for the output HTML.

After the scenario bundle is generated from Clementine, it is published to Cleo within the SPSS Web Deployment Framework. Using a Web browser, users can request scores or predictions based on individual records or data files that they specify.

**Modifying an Existing Bundle**

To modify a scenario bundle, you can open the bundle using the Cleo Wizard and make any changes on the Wizard pages. To apply your new specifications, republish or save the bundle.

**Cleo Stream Prerequisites**

To optimize use of the Cleo Wizard and ensure that your stream is prepared for deployment, consider the following recommendations before proceeding with the Wizard:

- Instantiate the data by clicking Read Values on the Type tab in each of the Source nodes. This optimizes the Wizard by making values available to you when specifying metadata.
- Terminate the stream with a Publisher node and perform a test execute to ensure that the stream is fully functional.

**Step 1: Cleo Wizard Overview Screen**

When you first open the Cleo Wizard, a welcome screen appears orienting you to the process of bundling the necessary stream components.
The rest of the Cleo Wizard pages walk you through the process of generating a bundle for deployment into the SPSS Web Deployment Framework. Help is provided on each screen, orienting you to the given task. For additional hints on each screen, click the Help button to open the relevant topic in the online Help.
Output Nodes

Overview of Output Nodes

Output nodes provide the means to obtain information about your data and models. Of course, this is important in every stage of data mining, from Business Understanding to Deployment. Output nodes also provide a mechanism for exporting data in various formats to interface with your other software tools.

There are 15 Output nodes:

- Table
- Matrix
- Analysis
- Data Audit
- Statistics
- Quality
- Report
- Set Globals
- Publisher
- Database Output
- Flat File
You can easily retrieve and manage charts, graphs, and tables generated in Clementine. The right side of the Clementine window contains the managers tool that helps you navigate the current output objects.

To view the managers tool, select Managers from the View menu. Then click the Outputs tab.

From this window, you can:

- Open existing output objects, such as histograms, evaluation charts, and tables.
- Save output objects.
- Add output files to the current project.
- Delete unsaved output objects from the current session.
- Rename output objects.

Right-click anywhere on the Outputs tab to access these options.
Output Browser Menus

Output browsers are used to display the results of output. The following menus are available in most output browsers.

**File menu.** The File menu contains file- and print-related operations.
- **Save.** Saves the results as a Clementine output object (*.cou). If the output has not been previously saved, you will be prompted for a filename and location.
- **Save As.** Saves the results as a Clementine output object with a new name.
- **Close.** Closes the browser window.
- **Close and Delete.** Closes the browser window and permanently deletes the output from the Outputs tab. You can also click the wastebasket icon at the corner of a browser window to close and delete.
- **Header and Footer.** Allows you to change the page headers and footers for printing.
- **Page Setup.** Allows you to change the page settings for printing output.
- **Print Preview.** Displays a preview of the output as it will appear when printed.
- **Print.** Opens the Print dialog box to print the output.
- **Export.** Exports the output as if the output settings file type were set to the corresponding file type. For more information, see “Output Node Output Tab” on page 491.
- **Publish to Web.** Publishes the output to the SPSS Web Deployment Framework (SWDF). For more information, see “Publish to Web” on page 490.
- **Export HTML.** Exports the output as HTML to the specified file.

**Edit menu.** The Edit menu contains editing operations.
- **Copy.** Copies the selected output to the clipboard.
- **Copy All.** Copies all output to the clipboard.
- **Copy (inc. field names).** For tabular output, such as tables and matrices, copies the selected cells and the associated column and row headings to the clipboard.
- **Select All.** Selects all content in the browser window.
- **Clear selection.** Deselects all content.
**Generate menu.** The Generate menu allows you to generate new nodes based on the contents of the output browser. Generate options will vary depending on the type of output you are browsing. See the documentation for the specific output type you are browsing for details of node-generating options.

**Publish to Web**

By selecting Publish to Web, you can publish your output to the SPSS Web Deployment Framework (SWDF).

**Figure 14-3**
*Publish to Web dialog box*

![Publish to Web dialog box](image)

Enter the authentication information and URL required to access the SWDF server. By default, Save username and server URL to options is selected, so your settings will be saved, and will be automatically used the next time you use the Publish to Web feature. If you do not want to save the information you enter, deselect this option. For more information on publishing to the SWDF, see the SWDF help files or contact your SWDF administrator.

*Note:* If you want to export simple HTML files for use with a standard Web server, use the Export HTML option instead.
Output Node Output Tab

The Output tab lets you control where the result of executing the Output node will go, and its format. Select the desired output style:

**Output to screen** (the default). Creates an output object to view online. The output object will appear on the Outputs tab of the managers window when the Output node is executed.

**Output to file.** Saves the output to an external file when the node is executed. If you choose this option, enter a filename (or navigate to a directory and specify a filename using the File Chooser button), and select a file type.

- **Formatted (*.tab)**. This option generates a formatted text file containing the data values. This style is often useful for generating a plain-text representation of the information that can be imported into other applications. This option is available for the Table and Matrix nodes.
Chapter 14

- **Data (comma delimited) (*.dat)**. This option generates a comma-delimited text file containing the data values. This style is often useful as a quick way to generate a data file that can be imported into spreadsheets or other data analysis applications. This option is available for the Table and Matrix nodes.

- **Transposed (*.dat)**. This option generates a comma-delimited text file similar to the Data (comma delimited) option, except that this option writes out a transposed version of the data. In the transposed data, the rows represent fields and the columns represent records. This option is available for the Table and Matrix nodes.

*Note*: For large tables, the above options can be somewhat inefficient, especially when working with a remote server. In such cases, using a File Output node will provide much better performance. For more information, see “Flat File Node” on page 539.

- **HTML (*.html)**. This option writes HTML-formatted output to a file or files. For tabular output (from the Table or Matrix nodes), a set of HTML files contains a contents panel listing field names, and the data in an HTML table. The table may be split over multiple HTML files if the number of rows in the table exceeds the Lines per page specification. In this case, the contents panel contains links to all table pages and provides a means of navigating the table. For nontabular output, a single HTML file is created containing the results of the node.

*Note*: If the HTML output contains only formatting for the first page, adjust the Lines per page specification to include all output on a single page. Or, if the output template for nodes such as the Report node contains custom HTML tags, be sure you have specified Custom as the format type.

- **Text File (*.txt)**. This option generates a text file containing the output. This style is often useful for generating output that can be imported into other applications, such as word processors or presentation software.

**Format.** For the Report node, you can choose whether output is automatically formatted or formatted using HTML included in the template. Select Custom to allow HTML formatting in the template.

**Title.** For the Report node, you can specify optional title text that will appear at the top of the report output.
**Highlight inserted text.** For the Report node, select this option to highlight text generated by CLEM expressions in the Report template. For more information, see “Report Node Template Tab” on page 527. This option is not recommended when using Custom formatting.

**Lines per page.** For the Report node, specify a number of lines to include on each page during Auto formatting of the output report.

### Table Node

The Table node allows you to create a table from your data, which can either be displayed on the screen or written to a file. This is useful anytime that you need to inspect your data values or export them in an easily readable form.

### Table Node Settings Tab

![Table node Settings tab](image)

**Figure 14-5**

*Table node Settings tab*
Highlight records where. You can highlight records in the table by entering a CLEM expression that is true for the records to be highlighted. This option is enabled only when Output to screen is selected.

Output. Output options can be set here, similar to the settings found on the common Output node Output tab. For more information, see “Output Node Output Tab” on page 491.

Table Node Format Tab

Figure 14-6

Table node Format Tab

The Format tab shows the list of fields and formatting options for each field.

Field. This shows the name of the selected field.

Justify. Specifies how the values should be justified within the table column. The default setting is Auto, which left-justifies symbolic values and right-justifies numeric values. You can override the default by selecting left, right, or center.
**Width.** By default, column widths are automatically calculated based on the values of the field. To override the automatic width calculation, deselect the Auto Width option for the desired field(s) and enter a Manual Width in the next column.

**View current fields.** By default, the dialog box shows the list of currently active fields. To view the list of unused fields, select View unused fields settings instead.

**Context menu.** The context menu for this tab provides various selection and setting update options.

- **Select All.** Selects all fields.
- **Select None.** Clears the selection.
- **Select Fields.** Selects fields based on type or storage characteristics. Options are Select Discrete, Select Range (numeric), Select Typeless, Select Strings, Select Numbers, or Select Date/Time. For more information, see “Data Types” in Chapter 7 on page 150.
- **Set Justify.** Sets the justification for the selected field(s). Options are Auto, Center, Left, or Right.
- **Set Auto Width.** Sets the use of automatic width calculation for the selected fields. Options are On or Off.
- **Set Manual Width.** Sets the manual field width for the selected field(s) (and disables Auto Width for those fields if necessary). Options include multiples of 5, up to 30. To set a manual width that is not a multiple of 5, edit the value directly in the table.
The table browser displays tabular data and allows you to perform operations on the table, including selecting cells, copying the table to the clipboard in whole or in part, generating new nodes based on the table selection, searching the table, and saving and printing the table.

**Selecting cells.** To select a single cell, click on that cell. To select a rectangular range of cells, click on one corner of the desired range, drag the mouse to the other corner of the range, and release the mouse button. To select an entire column, click on the column heading. Shift-click or Ctrl-click on column headings to select multiple columns.

When you make a new selection, the old selection is cleared. By holding down the Ctrl key while selecting, you can add the new selection to any existing selection instead of clearing the old selection. You can use this method to select multiple, noncontiguous regions of the table. There are also selection options available on the Edit menu.
Reordering columns. You can move columns in the table by clicking the heading of the column you want to move and dragging it to the desired location. As you drag the column, the table updates to show new order of columns. You can move only one column at a time.

Searching the table. The search button (with the binoculars icon) on the main toolbar activates the search toolbar, allowing you to search the table for specific values. You can search forward or backward in the table, you can specify a case-sensitive search (the Aa button), and you can interrupt a search-in-progress with the interrupt search button.

Figure 14-8
Table with search controls activated

Generating new nodes. The Generate menu contains node generation operations.

- Select Node ("Records"). Generates a Select node that selects the records for which any cell in the table is selected.

- Select ("And"). Generates a Select node that selects records containing all of the values selected in the table.
- **Select ("Or")**. Generates a Select node that selects records containing *any* of the values selected in the table.

- **Derive ("Records")**. Generates a Derive node to create a new flag field. The flag field contains “T” for records for which any cell in the table is selected and “F” for the remaining records.

- **Derive ("And")**. Generates a Derive node to create a new flag field. The flag field contains “T” for records containing *all* of the values selected in the table and “F” for the remaining records.

- **Derive ("Or")**. Generates a Derive node to create a new flag field. The flag field contains “T” for records containing *any* of the values selected in the table and “F” for the remaining records.

**Matrix Node**

The Matrix node allows you to create a table that shows relationships between fields. It is most commonly used to show the relationship between two symbolic fields, but it can also be used to show relationships between flag fields or between numeric fields.

**Matrix Node Settings Tab**

The Settings tab lets you specify options for the structure of the matrix.
Fields. Select a field selection type from the following options:

- **Selected.** This option allows you to select a symbolic field for the Rows and one for the Columns of the matrix. The rows and columns of the matrix are defined by the list of values for the selected symbolic field. The cells of the matrix contain the summary statistics selected below.

- **All flags (true values).** This option requests a matrix with one row and one column for each flag field in the data. The cells of the matrix contain the counts of double positives for each flag combination. In other words, for a row corresponding to *bought bread* and a column corresponding to *bought cheese*, the cell at the intersection of that row and column contains the number of records for which both *bought bread* and *bought cheese* are true.

- **All numerics.** This option requests a matrix with one row and one column for each numeric field. The cells of the matrix represent the sum of the cross-products for the corresponding pair of fields. In other words, for each cell in the matrix, the values for the row field and the column field are multiplied for each record and then summed across records.
**Cell contents.** If you have chosen Selected fields above, you can specify the statistic to be used in the cells of the matrix. Select a count-based statistic, or select an overlay field to summarize values of a numeric field based on the values of the row and column fields.

- **Cross-tabulations.** Cell values are counts and/or percentages of how many records have the corresponding combination of values. You can specify which cross-tabulation summaries you want using the options on the Appearance tab. For more information, see “Matrix Node Appearance Tab” on page 500.

- **Function.** If you select a summary function, cell values are a function of the selected overlay field values for cases having the appropriate row and column values. For example, if the row field is Region, the column field is Product, and the overlay field is Revenue, then the cell in the Northeast row and the Widget column will contain the sum (or average, minimum, or maximum) of revenue for widgets sold in the northeast region. The default summary function is Mean. You can select another function for summarizing the function field. Options include Mean, Sum, SDev (standard deviation), Max (maximum), or Min (minimum).

**Matrix Node Appearance Tab**

The Appearance tab allows you to control sorting and highlighting options for the matrix, as well as statistics presented for cross-tabulation matrices.
Rows and columns. Controls the sorting of row and column headings in the matrix. The default is Unsorted. Select Ascending or Descending to sort row and column headings in the specified direction.

Overlay. Allows you to highlight extreme values in the matrix. Values are highlighted based on cell counts (for cross-tabulation matrices) or calculated values (for function matrices).

- **Highlight top.** You can request the highest values in the matrix to be highlighted (in red). Specify the number of values to highlight.

- **Highlight bottom.** You can also request the lowest values in the matrix to be highlighted (in green). Specify the number of values to highlight.

**Note:** For the two highlighting options, ties can cause more values than requested to be highlighted. For example, if you have a matrix with six zeros among the cells and you request Highlight bottom 5, all six zeros will be highlighted.
Cross-tabulation cell contents. You can specify the summary statistics contained in the matrix for cross-tabulation matrices. This option is not available when using All flags (true values) of All Numerics.

- **Counts.** Cells include the number of records with the row value that have the corresponding column value. This is only default cell content.

- **Expected values.** Cells include the expected value for number of records in the cell, assuming that there is no relationship between the rows and columns. Expected values are based on the following formula:

\[ p(\text{row value}) \times p(\text{column value}) \times \text{total number of records} \]

- **Percentage of row.** Cells include the percentage of all records with the row value that have the corresponding column value. Percentages sum to 100 within rows.

- **Percentage of column.** Cells include the percentage of all records with the column value that have the corresponding row value. Percentages sum to 100 within columns.

- **Percentage of total.** Cells include the percentage of all records having the combination of column value and row value. Percentages sum to 100 over the whole matrix.

- **Include row and column totals.** Adds a row and a column to the matrix for column and row totals.

**Matrix Output Browser**

The matrix browser displays cross-tabulated data and allows you to perform operations on the matrix, including selecting cells, copying the matrix to the clipboard in whole or in part, generating new nodes based on the matrix selection, and saving and printing the matrix.
The usual saving, exporting, and printing options are available from the File menu, and the usual editing options are available from the Edit menu. For more information, see “Output Browser Menus” on page 489.

**Generate menu.** The Generate menu contains node generation operations. These operations are available only for cross-tabulated matrices, and you must have at least one cell selected in the matrix.

- **Select Node.** Generates a Select node that selects the records that match any selected cell in the matrix.

- **Derive Node (Flag).** Generates a Derive node to create a new flag field. The flag field contains “T” for records that match any selected cell in the matrix and “F” for the remaining records.

- **Derive Node (Set).** Generates a Derive node to create a new set field. The set field contains one category for each contiguous set of selected cells in the matrix.

**Selecting cells.** To select a single cell, click on that cell. To select a rectangular range of cells, click on one corner of the desired range, drag the mouse to the other corner of the range, and release the mouse button. To select an entire column, click on the column heading.
When you make a new selection, the old selection is cleared. By holding down the Ctrl key while selecting, you can add the new selection to any existing selection instead of clearing the old selection. You can use this method to select multiple, noncontiguous regions of the matrix. You can add contiguous regions to your selection by holding down the Shift key while selecting. Selection options are also available on the Edit menu.

**Analysis Node**

The Analysis node allows you to analyze predictive models to evaluate their ability to generate accurate predictions. Analysis nodes perform various comparisons between predicted values and actual values for one or more generated model nodes. Analysis nodes can also be used to compare predictive models to other predictive models.

When you execute an Analysis node, a summary of the analysis results is automatically added to the Analysis section on the Summary tab for each generated model node in the executed stream. The detailed analysis results appear on the Outputs tab of the managers window or can be written directly to a file.

**Analysis Node Analysis Tab**

The Analysis tab allows you to specify the details of the analysis.
Coincidence matrices (for symbolic targets). Shows the pattern of matches between each generated (predicted) field and its target field for symbolic targets. A table is displayed with rows defined by actual values and columns defined by predicted values, with the number of records having that pattern in each cell. This is useful for identifying systematic errors in prediction. If there is more than one generated field related to the same output field but produced by different models, the cases where these fields agree and disagree are counted and the totals are displayed. For the cases where they agree, another set of correct/wrong statistics is displayed.

Performance evaluation. Shows performance evaluation statistics for models with symbolic outputs. This statistic, reported for each category of the output field(s), is a measure of the average information content (in bits) of the model for predicting records belonging to that category. It takes the difficulty of the classification problem into account, so accurate predictions for rare categories will earn a higher performance evaluation index than accurate predictions for common categories. If the model does no better than guessing for a category, the performance evaluation index for that category will be 0.
**Confidence figures (if available).** For models that generate a confidence field, this option reports statistics on the confidence values and their relationship to predictions. There are two settings for this option:

- **Threshold for.** Reports the confidence level above which the accuracy will be the specified percentage.

- **Improve accuracy.** Reports the confidence level above which the accuracy is improved by the specified factor. For example, if the overall accuracy is 90% and this option is set to 2.0, the reported value will be the confidence required for 95% accuracy.

**User defined analysis.** You can specify your own analysis calculation to be used in evaluating your model(s). Use CLEM expressions to specify what should be computed for each record and how to combine the record-level scores into an overall score. Use the functions `@TARGET` and `@PREDICTED` to refer to the target (actual output) value and the predicted value, respectively.

- **If.** Specify a conditional expression if you need to use different calculations depending on some condition.

- **Then.** Specify the calculation if the If condition is true.

- **Else.** Specify the calculation if the If condition is false.

- **Use.** Select a statistic to compute an overall score from the individual scores.

**Break down analysis by fields.** Shows the symbolic fields available for breaking down the analysis. In addition to the overall analysis, a separate analysis will be reported for each category of each breakdown field.

**Analysis Output Browser**

The analysis output browser lets you see the results of executing the Analysis node. The usual saving, exporting, and printing options are available from the File menu. For more information, see “Output Browser Menus” on page 489.
When you first browse Analysis output, the results are expanded. To hide results after viewing them, use the expander control to the left of the item to collapse the specific results you want to hide, or click the Collapse All button to collapse all results. To see results again after collapsing them, use the expander control to the left of the item to show the results, or click the Expand All button to show all results.

**Results for output field.** The Analysis output contains a section for each output field for which there is a corresponding prediction field created by a generated model.
Comparing. Within the output field section is a subsection for each prediction field associated with that output field. For symbolic output fields, the top level of this section contains a table showing the number and percentage of correct and incorrect predictions and the total number of records in the stream. For numeric output fields, this section shows the following information:

- **Minimum Error.** Shows the minimum error (difference between observed and predicted values).
- **Maximum Error.** Shows the maximum error.
- **Mean Error.** Shows the average (mean) of errors across all records. This indicates whether there is a systematic bias (a stronger tendency to overestimate than to underestimate, or vice versa) in the model.
- **Mean Absolute Error.** Shows the average of the absolute values of the errors across all records. Indicates the average magnitude of error, independent of the direction.
- **Standard Deviation.** Shows the standard deviation of the errors.
- **Linear Correlation.** Shows the linear correlation between the predicted and actual values. This statistic varies between –1.0 and 1.0. Values close to +1.0 indicate a strong positive association, such that high predicted values are associated with high actual values and low predicted values are associated with low actual values. Values close to –1.0 indicate a strong negative association, such that high predicted values are associated with low actual values, and vice versa. Values close to 0.0 indicate a weak association, such that predicted values are more or less independent of actual values.
- **Occurrences.** Shows the number of records used in the analysis.

**Coincidence Matrix.** For symbolic output fields, if you requested a coincidence matrix in the analysis options, a subsection appears here containing the matrix. The rows represent actual observed values, and the columns represent predicted values. The cell in the table indicates the number of records for each combination of predicted and actual values.

**Performance Evaluation.** For symbolic output fields, if you requested performance evaluation statistics in the analysis options, the performance evaluation results appear here. Each output category is listed with its performance evaluation statistic.
Confidence Values Report. For symbolic output fields, if you requested confidence values in the analysis options, the values appear here. The following statistics are reported for model confidence values:

- **Range.** Shows the range (smallest and largest values) of confidence values for records in the stream data.

- **Mean Correct.** Shows the average confidence for records that are classified correctly.

- **Mean Incorrect.** Shows the average confidence for records that are classified incorrectly.

- **Always Correct Above.** Shows the confidence threshold above which predictions are always correct and the percentage of cases meeting this criterion.

- **Always Incorrect Below.** Shows the confidence threshold below which predictions are always incorrect and the percentage of cases meeting this criterion.

- **X% Accuracy Above.** Shows the confidence level at which accuracy is $X\%$. $X$ is approximately the value specified for Threshold for in the Analysis options. For some models and data sets, it is not possible to choose a confidence value that gives the exact threshold specified in the options (usually due to clusters of similar cases with the same confidence value near the threshold). The threshold reported is the closest value to the specified accuracy criterion that can be obtained with a single confidence value threshold.

- **X Fold Correct Above.** Shows the confidence value at which accuracy is $X$ times better than it is for the overall data set. $X$ is the value specified for Improve accuracy in the Analysis options.

Agreement between. If two or more generated models that predict the same output field are included in the stream, you will also see statistics on the agreement between predictions generated by the models. This includes the number and percentage of records for which the predictions agree (for symbolic output fields) or error summary statistics (for numeric output fields). For symbolic fields, it includes an analysis of predictions compared to actual values for the subset of records on which the models agree (generate the same predicted value).
Data Audit Node

The Data Audit node provides a comprehensive first-look at the data you bring into Clementine. Often used during the initial data exploration, the data audit report shows summary statistics as well as histograms and distribution graphs for each data field. The results are displayed in an easy-to-read matrix that can be sorted and used to generate full-size graphs and data preparation nodes.

To use the Data Audit node, simply attach the node to a data source and execute it. Or, for more information on data types, add the Data Audit node downstream of an instantiated Type node.

Figure 14-14
Typical use of the Data Audit node in a stream

You can double-click the Data Audit node in the stream to specify fields for auditing as well as an overlay field for graphs. Because an initial audit is particularly effective when dealing with “big data,” a sampling option is available to reduce processing time during the initial exploration.

When you execute a Data Audit node, an audit report is created for the selected fields. Similar to other Output nodes, the audit results appear on the Outputs tab of the managers window or can be written directly to a file.

Data Audit Node Settings Tab

The Settings tab allows you to specify the parameters for the data audit performed on your data.
Selecting Fields

**Default.** By default, the Data Audit node creates a report for all fields based upon settings in the Type node. This enables quick usage of the Data Audit node. For example, you can simply attach the node to your stream and click Execute to generate an audit report for all fields. The default behavior is as follows:

- If there are no Type node settings, all fields are included in the report.
- If there are Type settings (regardless of whether or not they are instantiated) all IN, OUT, and BOTH fields are included in the display. If there is a single OUT field use it as the Overlay field. If there is more than one OUT field specified, a default overlay not specified.

**Use custom fields.** Select this option to manually specify fields for inclusion in the audit. Use the field chooser button on the right to select fields individually or by type.
Fields. This box contains all fields manually selected for inclusion in the audit. Remove fields using the X button to the right of the field list box.

Overlay. To customize the audit report, you can manually select a field to use as an overlay for thumbnail graphs. By default, an overlay is selected based upon settings in the Type node as specified above.

Setting Display Options

Using options in the Display section of the Settings tab, you can select statistics and graphs for inclusion in the data audit report. To exclude statistics from the report, deselect Basic Statistics and Median and Mode. This produces a matrix containing only field name and thumbnail graphs.

Graphs. Select this option to view the graphs for fields selected on the Settings tab.

Basic statistics. Select this option to view the following statistics for fields selected on the Settings tab.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Availability by Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation Coefficient</td>
<td>Range fields (when range overlay)</td>
</tr>
<tr>
<td>Min</td>
<td>All numeric fields</td>
</tr>
<tr>
<td>Max</td>
<td>All numeric fields</td>
</tr>
<tr>
<td>Mean</td>
<td>Range fields</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>Range fields</td>
</tr>
<tr>
<td>Skewness</td>
<td>Range fields</td>
</tr>
<tr>
<td>Unique</td>
<td>Set, Flag fields</td>
</tr>
<tr>
<td>Valid</td>
<td>All fields</td>
</tr>
</tbody>
</table>

Median and mode. Select this option to view Median, Mode, Unique, and Valid columns for fields selected on the Settings tab.
Data Audit Node Sample Tab

The Sample tab allows you to specify when data will be sampled for the audit. For large data sets, sampling reduces processing time while providing an accurate initial picture of the data.

Figure 14-16
Data Audit node Sample tab

Use automatic sampling criteria. Select to sample a random percentage of the data based upon the following criteria:
- If there are less than 250 fields, the automatic sample size will be 2,000 records.
- If there are more than 250 fields, the sample size will be 1,000 records.

You can also specify a seed value using the Set random seed control.

Sample when records greater than. Select to set custom settings for sampling. By default, records will be sampled at 2,000. Setting this number higher may increase processing time when rendering the data audit report.

Set random seed. When sampling is selected above, you can use this control to set a random seed and specify the seed value. Specifying a seed value allows you to reproduce the same list of randomly selected records if needed. Click the Generate button to generate a random seed.
Chapter 14

Note: Random sampling will automatically occur for all output when you request statistics that are based on sorting (for example, the median). Data that have been sampled for the audit will labeled “Sample” in the output.

Data Audit Output Browser

The output browser lets you see the results of executing the Data Audit node. The usual saving, exporting, and printing options are available from the File menu. For more information, see “Output Browser Menus” on page 489.

Figure 14-17
Data audit report in the output browser

The Data Audit report window is a powerful tool that enables you to compare fields quickly using thumbnail graphs and view a variety of statistics for all fields. Using your mouse to explore the graph and toolbar options, you can also:

- View values and ranges for fields by double-clicking a field in the Type or Unique columns.
- Sort columns by clicking on the column header to activate an arrow indicating sort order. This option is available for all columns in the report.
- View graph labels for set fields by hovering with the mouse over a bar to display its label in a tooltip.

- Generate full-size graphs from the thumbnail graph in the report. For more information, see “Generating Graphs and Nodes from the Audit” on page 516.

- Generate various nodes used in transforming and preparing data for analysis. For more information, see “Generating Graphs and Nodes from the Audit” on page 516.

- Export the report in HTML. For more information, see “Output Browser Menus” on page 489.

**Types of Results**

The audit results vary depending upon the type and presence of an overlay field. Overlay fields are automatically set based upon options in an upstream Type node or custom options specified in the Data Audit node dialog box.

If no overlay is selected, all charts are either bar charts (set or flag) or histograms (range). Note that typeless fields are not displayed.

**Figure 14-18**

*Excerpt of audit results without an overlay field*

<table>
<thead>
<tr>
<th>Field</th>
<th>Graph</th>
<th>Type</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Std. Dev</th>
<th>Skewness</th>
<th>Unique</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>Set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>300</td>
</tr>
<tr>
<td>rainfall</td>
<td>Range</td>
<td></td>
<td>15</td>
<td>100</td>
<td>88.18</td>
<td>27.81</td>
<td>-0.18</td>
<td></td>
<td>300</td>
</tr>
</tbody>
</table>

For a set or flag field overlay, the graphs are colored by the values of the overlay.

**Figure 14-19**

*Excerpt of audit results with a set field overlay*

<table>
<thead>
<tr>
<th>Field</th>
<th>Graph</th>
<th>Type</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Std. Dev</th>
<th>Skewness</th>
<th>Unique</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>Set</td>
<td></td>
<td></td>
<td>30</td>
<td>70.17</td>
<td>-0.06</td>
<td></td>
<td>4</td>
<td>300</td>
</tr>
<tr>
<td>rainfall</td>
<td>Range</td>
<td></td>
<td>15</td>
<td>100</td>
<td>88.18</td>
<td>27.81</td>
<td>-0.18</td>
<td></td>
<td>300</td>
</tr>
</tbody>
</table>

*Note:* If an overlay set has more than 100 values, a warning is raised and the overlay is not included.
For a scale field overlay, two-dimensional scatterplots are generated rather than one-dimensional bars and histograms. In this case, the $x$ axis maps to the overlay field, enabling you to see the same scale on all $x$ axes as you read down the table.

**Figure 14-20**
Excerpt of audit results with a scale field overlay

<table>
<thead>
<tr>
<th>Field</th>
<th>Graph</th>
<th>Type</th>
<th>Correlation</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Std. Dev</th>
<th>Skewness</th>
<th>Unique</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td></td>
<td>Bar</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>4</td>
<td>300</td>
</tr>
<tr>
<td>rainfall</td>
<td></td>
<td>Range</td>
<td>0.56</td>
<td>15</td>
<td>100</td>
<td>66.10</td>
<td>27.31</td>
<td>-0.16</td>
<td>--</td>
<td>300</td>
</tr>
</tbody>
</table>

**Generating Graphs and Nodes from the Audit**

The audit report provides a useful starting point for your data preparation. Based upon the initial graphs and summary statistics, you might decide to recode a numeric field, derive a new field, or reclassify the values of a set field. Or, you may want to explore further using more sophisticated visualization. This functionality is available directly from the audit report using the Generate menu to create nodes and graphs based upon selections in the audit browser.

**Figure 14-21**
Generate menu in the browser window

**Generating Graphs**

When one or more fields is selected in the browser window, you can generate a graph node for the type of thumbnail graph shown, or you can generate full-size graph output.

**To generate a graph node:**
Select one or more fields in the browser.

From the Generate menu, select Graph node.

Open the graph node added to stream canvas to specify chart options and custom overlays.

To generate a graph (output window only):

Double-click a graph in the browser.

or

Select a single field in the browser.

From the Generate menu, select Graph output.

For range fields, a histogram is added to the Outputs tab of the managers window and opened for viewing. For discrete fields, a distribution is added to the Outputs tab and opened for viewing.

Note: If a thumbnail graph was based upon sampled data, the generated graph will contain all cases if the original data stream is still open.

Generating Nodes for Data Preparation

A variety of nodes used in data preparation can be generated directly from the audit report browser. For example:

- You can derive a new field based upon the values of `claimvalue` and `farmincome` by selecting both in the audit report and choosing Derive from the Generate menu. The new node is added to the stream canvas.

- Similarly, you may determine, based upon audit results, that recoding `farmincome` into percentile-based bins provides more focused analysis. To generate a Binning node, select the field row in the display and choose Binning from the Generate menu.

The following Field Operation nodes may be generated:

- Filter
- Derive
chapter 14

- Binning
- Reclassify

Once a node is generated and added to the stream canvas, you must attach it to the stream and open the node to specify options for the selected field(s).

Statistics Node

The Statistics node gives you basic summary information about numeric fields. You can get summary statistics for individual fields and correlations between fields.

Statistics Node Settings Tab

Figure 14-22

Statistics node Settings tab
**Examine.** Select the field or fields for which you want individual summary statistics. You can select multiple fields.

**Statistics.** Select the statistics to report. Available options include Count, Mean, Sum, Min, Max, Range, Variance, Std Dev, Std Error of Mean, Median, and Mode.

**Correlate.** Select the field or fields that you want to correlate. You can select multiple fields. When correlation fields are selected, the correlation between each Examine field and the correlation field(s) will be listed in the output.

**Correlation Labels.** You can customize the descriptive labels attached to correlation values in the output.

---

**Statistics Node Correlation Labels**

Clementine can characterize correlations with descriptive labels to help highlight important relationships. By default, correlations between 0.0 and 0.3333 (in absolute value) are labeled as *Weak*, those between 0.3333 and 0.6666 are labeled as *Medium*, and those between 0.6666 and 1.0 are labeled as *Strong*. Because the way you characterize correlation values depends greatly on the problem domain, you may want to customize the ranges and labels to fit your specific situation.

**Figure 14-23**

*Correlation Labels dialog box*

![Correlation Labels dialog box](image)

**Show correlation strength labels in output.** This option is selected by default. Deselect this option to omit the descriptive labels from the output.

**Define correlation value ranges and labels.** To change the ranges that define the categories, enter the new threshold(s) or use the spin controls to change the values. To change the labels used for each range, enter the new text label in the Label text box.
Chapter 14

Statistics Output Browser

The statistics output browser displays the results of the statistical analysis and allows you to perform operations, including selecting fields, generating new nodes based on the selection, and saving and printing the results. The usual saving, exporting, and printing options are available from the File menu, and the usual editing options are available from the Edit menu. For more information, see “Output Browser Menus” on page 489.

When you first browse Statistics output, the results are expanded. To hide results after viewing them, use the expander control to the left of the item to collapse the specific results you want to hide, or click the Collapse All button to collapse all results. To see results again after collapsing them, use the expander control to the left of the item to show the results, or click the Expand All button to show all results.

Figure 14-24
Statistics output browser

The output contains a section for each Examine field, containing a table of the requested statistics.
- **Count.** The number of records with valid values for the field.
- **Mean.** The average (mean) value for the field across all records.
- **Sum.** The sum of values for the field across all records.
- **Min.** The minimum value for the field.
- **Max.** The maximum value for the field.
- **Range.** The difference between the minimum and maximum values.
- **Variance.** A measure of the variability in the values of a field. It is calculated by taking the difference between each value and the overall mean, squaring it, summing across all of the values, and dividing by the number of records.
- **Standard Deviation.** Another measure of variability in the values of a field, calculated as the square root of the variance.
- **Standard Error of Mean.** A measure of the uncertainty in the estimate of the field's mean if the mean is assumed to apply to new data.
- **Median.** The “middle” value for the field; that is, the value that divides the upper half of the data from the lower half of the data (based on values of the field).
- **Mode.** The most common single value in the data.

If you specified correlate fields, the output also contains a section listing the Pearson correlation between the Examine field and each correlate field, and optional descriptive labels for the correlation values. The **correlation** measures the strength of relationship between two numeric fields. It takes values between –1.0 and 1.0. Values close to +1.0 indicate a strong positive association, such that high values on one field are associated with high values on the other, and low values are associated with low values. Values close to –1.0 indicate a strong negative association, so that high values for one field are associated with low values for the other, and vice versa. Values close to 0.0 indicate a weak association, so that values for the two fields are more or less independent.

**Generate menu.** The Generate menu contains node generation operations.
- **Filter.** Generates a Filter node to filter out fields that are uncorrelated or weakly correlated with other fields.
Generating a Filter Node from Statistics

The Filter node generated from a Statistics output browser will filter fields based on their correlations with other fields. It works by sorting the correlations in order of absolute value, taking the largest correlations (according to the criterion set in the dialog box), and creating a filter that passes all fields that appear in any of those large correlations.

**Mode.** Decide how to select correlations. Include causes fields appearing in the specified correlations to be retained. Exclude causes the fields to be filtered.

**Include/Exclude fields appearing in.** Define the criterion for selecting correlations.

- **Top number of correlations.** Selects the specified number of correlations and includes/excludes fields that appear in any of those correlations.
- **Top percentage of correlations (%).** Selects the specified percentage (n%) of correlations and includes/excludes fields that appear in any of those correlations.
- **Correlations greater than.** Selects correlations greater in absolute value than the specified threshold.

Quality Node

The Quality node reports on the quality of your data by checking for missing values or blanks. The node can take Clementine blank definitions into account or deal with empty or white space values. From the Quality output browser you can generate Select or Filter nodes based on various data quality characteristics.
To analyze the quality of all of your data, select **Evaluate all fields**. To analyze only certain fields, select **Evaluate selected fields** and select the fields of interest.

**Treat as invalid values.** Select the data features that you want to consider as invalid values.

- **Null (undefined) value.** Considers system ($null$) values as invalid.
- **Empty string.** Consider empty strings as invalid.
- **White space.** Consider values that contain only white space (spaces, tabs, or new lines) as invalid.
- **Blank values.** Consider blank values as defined by an upstream Type node or Source node as invalid. For more information, see “Specifying Missing Values” in Chapter 9 on page 224.

**Calculate.** Select calculation options for the quality report.
- **Count of records with valid values.** Select this option to show the number of records with valid values for each evaluated field.
- **Breakdown counts of records with invalid values.** Select this option to show the number of records with each type of invalid value for each field.

**Quality Node Output Browser**

*Figure 14-27  
Quality browser window*

<table>
<thead>
<tr>
<th>Field</th>
<th>% Complete</th>
<th>Valid Records</th>
<th>Null Value</th>
<th>Empty String</th>
<th>White Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>100</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>region</td>
<td>100</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>house size</td>
<td>100</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rainfall</td>
<td>100</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>land quality</td>
<td>82.33</td>
<td>247</td>
<td>53</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>farm income</td>
<td>76</td>
<td>234</td>
<td>68</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>main crop</td>
<td>100</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>claim type</td>
<td>100</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>claim value</td>
<td>100</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The quality browser displays the results of the data quality analysis and allows you to perform operations, including selecting fields, generating new nodes based on the selection, and saving and printing the results.

**Quality results.** The data quality report lists the fields in descending order of completeness. The fields with the highest data quality (the lowest proportion of invalid values as defined in the options) are listed at the top, and those with the lowest data quality (the highest proportion of invalid values) are listed at the bottom.

**Selecting fields.** You can select fields by clicking directly on them in the list. Multiple fields can be selected by holding down the Shift key (to add contiguous fields) or the Ctrl key (to add noncontiguous fields) while clicking. You can deselect a selected field by clicking it while pressing the Ctrl key or by simply selecting another field. You can also use the options on the Edit menu to select all fields or to clear the selection.
**Generate menu.** The Generate menu contains node generation operations.

- **Filter.** Generates a Filter node that filters fields based on the results of the Quality analysis. For more information, see “Generating a Filter Node from Quality Analysis” on page 525.

- **Select.** Generates a Select node that selects the records based on the results of the Quality analysis. For more information, see “Generating a Select Node from Quality Analysis” on page 525.

**Generating a Filter Node from Quality Analysis**

After executing a Quality node, you can create a new Filter node based on the results of the Quality analysis.

![Generate Filter from Quality dialog box](image)

- **Mode.** Select the desired operation for specified fields, either Include or Exclude.

- **Selected fields.** The Filter node will include/exclude the fields selected in the Quality output table. If no fields are selected in the table, no fields will be used for the operation.

- **Fields with quality percentage higher than.** The Filter node will include/exclude fields where the percentage of complete records is greater than the specified threshold. The default threshold is 50%.

**Generating a Select Node from Quality Analysis**

After executing a Quality node, you can create a new Select node based on the results of the Quality analysis.
**Chapter 14**

**Figure 14-29**
Generate Select node dialog box

![Select node dialog box](image)

**Select when record is.** Specify whether records should be kept when they are Valid or Invalid.

**Look for invalid values in.** Specify where to check for invalid values.
- **All fields.** The Select node will check all fields for invalid values.
- **Fields selected in table.** The Select node will check only the fields currently selected in the Quality output table.
- **Fields with quality percentage higher than.** The Select node will check fields where the percentage of complete records is greater than the specified threshold. The default threshold is 50%.

**Consider a record invalid if an invalid value is found in.** Specify the condition for identifying a record as invalid.
- **Any of the above fields.** The Select node will consider a record invalid if any of the fields specified above contains an invalid value for that record.
- **All of the above fields.** The Select node will consider a record invalid only if all of the fields specified above contain invalid values for that record.
Report Node

The Report node allows you to create formatted reports containing fixed text as well as data and other expressions derived from the data. You specify the format of the report by using text templates to define the fixed text and the data output constructions. You can provide custom text formatting using HTML tags in the template and by setting options on the Output tab. Data values and other conditional output are included in the report using CLEM expressions in the template.

Report Node Template Tab

Creating a template. To define the contents of the report, you create a template on the Report node Template tab. The template consists of lines of text, each of which specifies something about the contents of the report, and some special tag lines used to indicate the scope of the content lines. Within each content line, CLEM
expressions enclosed in square brackets ([ ]) are evaluated before the line is sent to the report. There are three possible scopes for a line in the template:

- **Fixed.** Lines that are not marked otherwise are considered fixed. Fixed lines are copied into the report only once, after any expressions that they contain are evaluated. For example, the line

This is my report, printed on [@TODAY]

would copy a single line to the report, containing the text and the current date.

- **Global (iterate ALL).** Lines contained between the special tags #ALL and # are copied to the report once for each record of input data. CLEM expressions (enclosed in brackets) are evaluated based on the current record for each output line. For example, the lines

#ALL
For record [@INDEX], the value of AGE is [AGE]
#

would include one line for each record indicating the record number and age.

- **Conditional (iterate WHERE).** Lines contained between the special tags #WHERE <condition> and # are copied to the report once for each record where the specified condition is true. The condition is a CLEM expression. (In the WHERE condition, the brackets are optional.) For example, the lines

#WHERE [SEX='M']
Male at record no. [@INDEX] has age [AGE].
#

will write one line to the file for each record with a value of “M” for sex.

The complete report will contain the fixed, global, and conditional lines defined by applying the template to the input data.

You can specify options for displaying or saving results using the Output tab, common to various types of Output nodes. For more information, see “Output Node Output Tab” on page 491.
Report Node Output Browser

The report browser shows you the contents of the generated report. The usual saving, exporting, and printing options are available from the File menu, and the usual editing options are available from the Edit menu. For more information, see “Output Browser Menus” on page 489.

Figure 14-31
Report browser

Set Globals Node

The Set Globals node scans the data and computes summary values that can be used in CLEM expressions. For example, you can use a Set Globals node to compute statistics for a field called age and then use the overall mean of age in CLEM expressions by inserting the function @GLOBAL_MEAN(age). For more information, see “CLEM Reference Overview” in Appendix A on page 663.
**Set Globals Node Settings Tab**

Figure 14-32
*Set Globals node Settings tab*

Globals to be created. Select the field or fields for which you want globals to be available. You can select multiple fields. For each field, specify the statistics to compute by making sure that the statistics you want are selected in the columns next to the field name.

- **MEAN.** The average (mean) value for the field across all records.
- **SUM.** The sum of values for the field across all records.
- **MIN.** The minimum value for the field.
- **MAX.** The maximum value for the field.
- **SDEV.** A measure of variability in the values of a field, calculated as the square root of the variance. For more information, see the Glossary.

Default operation(s). The options selected here will be used when new fields are added to the Globals list above. To change the default set of statistics, select or deselect statistics as appropriate. You can also use the Apply button to apply the default operations to all fields in the list.
Clear all globals before executing. Select this option to remove all global values before calculating new values. If this option is not selected, newly calculated values replace older values, but globals that are not recalculated remain available as well.

Display preview of globals created after execution. If you select this option, the Globals tab of the stream properties dialog box will appear after execution to display the calculated global values. For more information, see “Viewing Global Values for Streams” in Chapter 4 on page 84.

Solution Publisher Node

Clementine Solution Publisher is an add-on product that allows you to export entire Clementine streams in order to embed the streams in your own external applications. This allows you to interactively create your data mining process until you have a data modeling stream that serves your needs and then to use that stream in a production environment. Solution Publisher provides a more powerful deployment mechanism than exporting generated models because it publishes the entire stream, including data preprocessing done by the stream as well as the actual model used with the data.

Streams are exported as a published image file and an associated parameter file that can be executed using the Clementine Runtime. This removes much of the complexity associated with compilation of C code into an executable file.

Note: Solution Publisher is not part of the base Clementine program. To use Solution Publisher, you must have a license for it. If Solution Publisher is not enabled on your system, check with your system administrator to make sure all license codes have been activated, or contact SPSS about purchasing a Solution Publisher license.

For full details on how to use Solution Publisher, see the Solution Publisher User's Guide.pdf on the Clementine CD.
**Setting Options for the Solution Publisher Node**

Published name. Specify the name for the files to be published. Enter a filename or click the File Chooser button to browse to the file's location.

Export data. You can export records in several formats. Each format has its own associated options.

- Database Export Options. This option writes records to a database table. Database export options are the same as those for the Database Output node. For more information, see “Database Node Export Tab” on page 534.
Output Nodes

- **Flat file.** This option writes records to a delimited text file. File Export Options are the same as those for the Flat File Output node. For more information, see “Flat File Export Tab” on page 539.

- **SPSS file.** This option writes records to an SPSS data file. SPSS file export options are the same as those for the SPSS Export node, except that the application-launching options are not available in the Publisher node. For more information, see “SPSS Export Node Export Tab” on page 541.

- **SAS file.** This option writes records to a SAS data file. SAS file export options are the same as those for the SAS Export node. For more information, see “SAS Export Node Export Tab” on page 544.

When publishing to external applications, consider filtering extraneous fields or renaming fields to conform with input requirements. Both can be accomplished using the Filter tab in the Publisher dialog box. For more information, see “SPSS Import Node” in Chapter 5 on page 111.

**Database Output Node**

You can use Database nodes to write data to ODBC-compliant relational data sources.  

*Note:* To write to an ODBC data source, the data source must exist, and you must have write permission for that data source. Contact your database administrator if you have questions about creating or setting permissions for ODBC data sources.
**Database Node Export Tab**

Figure 14-34
*Database Output node Export tab*

![Database Output node Export tab]

**Data source.** Shows the selected data source. Enter the name or select it from the drop-down list. If you don’t see the desired database in the list, select Add new database connection and locate your database from the Database Connections dialog box. For more information, see “Adding a Database Connection” in Chapter 5 on page 108.

**Table name.** Enter the name of the table to which you want to send the data. If you select the Insert into table option, you can select an existing table in the database by clicking the Select button.

**Create table.** Select this option to create a new database table or to overwrite an existing database table.

**Insert into table.** Select this option to insert the data into an existing database table.

**Drop existing table.** Select this option to delete any existing table with the same name when creating a new table.

**Delete existing rows.** Select this option to delete existing rows from the table before exporting when inserting into a table.
Note: If either of the two options above are selected, you will receive an Overwrite warning message when you execute the node. To suppress the warnings, deselect Warn when a node overwrites a database table on the Notifications tab of User Options. For more information, see “Setting Notification Options” in Chapter 2 on page 33.

**Default string size.** Fields you have marked as typeless in an upstream Type node are written to the database as string fields. Specify the size of strings to be used for typeless fields.

**Quote table and column names.** Select options used when sending a CREATE TABLE statement to the database. Tables or columns with spaces or nonstandard characters must be quoted.

- **As needed.** Select to allow Clementine to automatically determine when quoting is needed on an individual basis.
- **Always.** Select to always enclose table and column names in quotes.
- **Never.** Select to disable the use of quotes.

**Generate an import node for this data.** Select to generate a Database source node for the data as exported to the specified data source and table. Upon execution, this node is added to the stream canvas.

Click the Schema button to open a dialog box where you can set SQL data types for your fields. For more information, see “Database Output Schema Options” on page 535.

Click the Advanced button to specify bulk loading and database commit options. For more information, see “Database Output Advanced Options” on page 536.

### Database Output Schema Options

The database output Schema dialog box allows you to set SQL data types for your fields. By default, Clementine will allow the database server to assign data types automatically. To override the automatic type for a field, find the row corresponding to the field and select the desired type from the drop-down list in the **Type** column of the schema table.
For types that take a length, precision, or scale argument (BINARY, VARBINARY, CHAR, VARCHAR, NUMERIC, and NUMBER), you should specify a length rather than allowing the database server to assign an automatic length. For example, specifying a sensible value, such as VARCHAR(25), for length ensures that the storage type in Clementine will be overwritten if that is your intention. To override the automatic assignment, select Specify from the Type drop-down list and replace the type definition with the desired SQL type definition statement.

The easiest way to do this is to first select the type that is closest to the desired type definition and then select Specify to edit that definition. For example, to set the SQL data type to VARCHAR(25), first set the type to VARCHAR(length) from the Type drop-down list, and then select Specify and replace the text length with the value 25.

**Database Output Advanced Options**

When you click the Advanced button from the Database and Publisher node dialog boxes, a new dialog box opens to specify technical details for exporting results to a database.
**Batch commit.** Select to turn off row-by-row commits to the database.

**Batch size.** Specify the number of records to send to the database before committing to memory. Lowering this number provides greater data integrity at the cost of slower transfer speeds. You may want to fine-tune this number for optimal performance with your database.

**Use bulk loading.** Select a method for bulk loading data to the database directly from Clementine.

- **Via ODBC.** Select to use the ODBC API to execute multiple-row inserts with greater efficiency than normal export to the database. Choose from row-wise or column-wise binding in the options below.

- **Via external loader.** Select to use a custom bulk loader program specific to your database. Selecting this option activates a variety of options below.

**Advanced ODBC Options.** These options are available only when Via ODBC is selected.
- **Row-wise.** Select row-wise binding to use the SQLBulkOperations call for loading data into the database. Row-wise binding typically improves speed compared to the use of parameterized inserts that insert data on a record-by-record basis.

- **Column-wise.** Select to use column-wise binding for loading data into the database. Column-wise binding improves performance by binding each database column (in a parameterized INSERT statement) to an array of \( N \) values. Executing the INSERT once causes \( N \) rows to be inserted into the database. This method can dramatically increase performance.

**External Loader Options.** When Via external loader is specified, a variety of options are displayed for exporting the dataset to a file and specifying and executing a custom loader program to load the data from that file into the database.

- **Use delimiter.** Specify which delimiter character should be used in the exported file. Select Tab to delimit with tab and Space to delimit with spaces. Select Other to specify another character, such as the comma (,).

- **Specify data file.** Select to enter the path to use for the data file written during bulk loading. By default, a temporary file is created in the temp directory on the server.

- **Specify loader program.** Select to specify a bulk loading program. By default, the software searches the /scripts subdirectory of the Clementine (client, server, and Solution Publisher) installation for a python script to execute for a given database. Several scripts have been included with the software. Check the /scripts subdirectory for available scripts and technical documentation.

- **Generate log.** Select to generate a log file to the specified directory. The log file contains error information and is useful if the bulk load operation fails.

- **Check table size.** Select to perform table checking that ensures that the increase in table size corresponds to the number of rows exported from Clementine.

- **Extra loader options.** Specify additional arguments to the loader program. Use double-quotes for arguments containing spaces.

  Double-quotes are included in optional arguments by escaping with a backslash. For example, the option specified as -comment "This is a \"comment\"" includes both the -comment flag and the comment itself rendered as This is a "comment".

  A single backslash can be included by escaping with another back-slash. For example, the option specified as -specialdir "C:\Test Scripts\" includes the flag -specialdir and the directory rendered as C:\Test Scripts\.
Flat File Node

The File node allows you to write data to a delimited text file. This is useful for exporting data that can be read by other analysis or spreadsheet software.

Note: You cannot write files in the old Clementine cache format, because Clementine no longer uses that format for cache files. Clementine cache files are now saved in SPSS .sav format, which you can write using an SPSS Export node. For more information, see “SPSS Export Node” on page 540.

Flat File Export Tab

Figure 14-38
File node Export tab

**Export file.** Specify the name of the file. Enter a filename or click the File Chooser button to browse to the file's location.
**Write mode.** If Overwrite is selected, any existing data in the specified file will be overwritten. If Append is selected, output from this node will be added to the end of the existing file, preserving any data it contains.

- **Include field names.** If this option is selected, field names will be written to the first line of the output file. This option is available only for the Overwrite write mode.

**New line after each record.** If this option is selected, each record will be written on a new line in the output file.

**Field separator.** Select the character to insert between field values in the generated text file. Options are Comma, Tab, Space, and Other. If you select Other, enter the desired delimiter character(s) in the text box.

**Symbol quotes.** Select the type of quoting to use for values of symbolic fields. Options are None (values are not quoted), Single ('), Double ("), and Other. If you select Other, enter the desired quoting character(s) in the text box.

**Decimal symbol.** Specify how decimals should be represented in the exported data.

- **Stream default.** The decimal separator defined by the current stream's default setting will be used. This will normally be the decimal separator defined by the machine's locale settings.
- **Period (.).** The period character will be used as the decimal separator.
- **Comma (.).** The comma character will be used as the decimal separator.

**Generate an import node for this data.** Select this option to automatically generate a Variable File source node that will read the exported data file. For more information, see “Variable File Node” in Chapter 5 on page 98.

---

**SPSS Export Node**

The SPSS Export node allows you to export data in SPSS .sav format. SPSS .sav files can be read by SPSS Base and other SPSS products. This is now also the format used for Clementine cache files.
Mapping Clementine field names to SPSS variable names can sometimes cause errors because SPSS variable names are limited to eight characters and cannot include certain characters, such as spaces, $, –, etc. There are two ways to adjust for these restrictions:

- You can rename fields conforming to SPSS variable name requirements by clicking the Filter tab. For more information, see “Renaming or Filtering Fields for Export” on page 542.

- The SPSS Export node provides an option to save Clementine field names as variable labels rather than as variable names to avoid this problem. (SPSS variable labels can be up to 255 characters and can contain any combination of characters.) If you choose this option, the variable names become VAR00001, VAR00002, etc., and each variable has its original Clementine field name as a variable label.

**SPSS Export Node Export Tab**

![SPSS Export Node Export Tab](image)
**Export file.** Specify the name of the file. Enter a file name or click the file chooser button to browse to the file's location.

**Export field names as variable.** To use the Clementine field names as variable names, select names. Clementine allows characters in field names that are illegal in SPSS variable names. To prevent possibly creating illegal SPSS names, select labels instead.

**Launch application.** If SPSS or AnswerTree is installed on your computer, you can invoke them directly on the saved data file. Select the program to open. If you have the applications installed but are having problems launching them, check your helper applications settings. For more information, see “Helper Applications” on page 550.

To simply create an SPSS `.sav` file without opening an external program, deselect this option.

**Generate an import node for this data.** Select this option to automatically generate an SPSS File node that will read the exported data file. For more information, see “SPSS Import Node” in Chapter 5 on page 111.

**Renaming or Filtering Fields for Export**

Before exporting or deploying data from Clementine to external applications such as SPSS, it may be necessary to rename or truncate field names. The SPSS Procedure, SPSS Export, and Publisher node dialog boxes contain a Filter tab to facilitate this process.

A basic description of Filter functionality is discussed elsewhere. For more information, see “Setting Filtering Options” in Chapter 7 on page 165. This topic provides tips for reading data into SPSS.
Figure 14-40
Renaming fields for SPSS on the Filter tab of the SPSS Procedure Node

Tips for SPSS

To automatically truncate field names to conform with SPSS, select Rename for SPSS from the Filter menu. This adjusts field names in the Filter window according to the following restrictions for data in SPSS version 12.0 and higher.

Table 14-2
Field name restrictions and corrective action

<table>
<thead>
<tr>
<th>SPSS Restriction</th>
<th>Corrective Renaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field names must begin with a letter.</td>
<td>The letter X is added to the beginning of the name.</td>
</tr>
<tr>
<td>The name cannot include blank spaces or any special characters except a period (.) or the symbols @, #, _, or $</td>
<td>Invalid characters are replaced with a # symbol.</td>
</tr>
<tr>
<td>Field names cannot end in a period.</td>
<td>Periods are replaced with a # symbol.</td>
</tr>
</tbody>
</table>
Chapter 14

**SPSS Restriction**

<table>
<thead>
<tr>
<th>SPSS Restriction</th>
<th>Corrective Renaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of field names cannot exceed 64 characters.</td>
<td>Long names are truncated to 64 characters, according to standards for SPSS 12.0 and higher.</td>
</tr>
<tr>
<td>Field names must be unique. <em>Note:</em> Names in SPSS are not case sensitive.</td>
<td>Duplicate names are truncated to 5 characters and then appended with an index ensuring uniqueness.</td>
</tr>
<tr>
<td>Reserved keywords are: ALL, NE, EQ, TO, LE, LT, BY, OR, GT, AND, NOT, GE, WITH</td>
<td>Fields names matching a reserved word are appended with the # symbol. For example, WITH becomes WITH#.</td>
</tr>
</tbody>
</table>

**SAS Export Node**

The SAS Export node allows you to write data in SAS format to be read into SAS or a SAS-compatible software package. You can export in three SAS file formats: SAS for Windows/OS2, SAS for UNIX, or SAS Version 7/8.

**SAS Export Node Export Tab**

**Figure 14-41**

*SAS Export node Export tab*

<table>
<thead>
<tr>
<th>Export file</th>
<th>SAS for Windows/OS2 (*.sd2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Export field names as variable:</td>
<td>names</td>
</tr>
<tr>
<td>Generate an import node for this data</td>
<td></td>
</tr>
</tbody>
</table>

**Export file.** Specify the name of the file. Enter a filename or click the File Chooser button to browse to the file's location.

Export field names as variable. To use the Clementine field names as variable names, select names. Clementine allows characters in field names that are illegal in SAS variable names. To prevent possibly creating illegal SAS names, select labels instead.

Generate an import node for this data. Select this option to automatically generate a SAS File node that will read the exported data file. For more information, see “SAS Import Node” in Chapter 5 on page 113.

Excel Node

The Excel node exports the data and opens it in Microsoft Excel as a spreadsheet. (If Excel is not installed, executing this node will report an error and save the file in a format readable by Excel).

If you have Excel installed but are having problems launching it, check your helper applications settings. For more information, see “Helper Applications” on page 550.

Excel Node Export Tab

Figure 14-42
Excel node Export Tab
Create file. Indicate where you want the export file to be created.

- In temporary directory. This option will create the file in the temporary directory, with an automatically generated filename.

- With specified file name. This option will save the file to the filename specified.

File name. If you select With specified file name above, specify the name of the file to be exported. Enter a filename or click the File Chooser button to browse to the file's location.

File type. Select the format for the exported file. Options are CSV (Comma delimited) or Text (tab delimited).

Decimal symbol. Specify how decimals should be represented in the exported data.

- Locale default. The decimal separator defined by the machine's locale setting will be used.

- Period (.). The period character will be used as the decimal separator.

- Comma (,). The comma character will be used as the decimal separator.

SPSS Procedure Node

If you have SPSS installed on your machine, the SPSS Procedure node allows you to call an SPSS procedure to analyze your Clementine data. You can view the results in a browser window or save results in the SPSS output file format. A wide variety of SPSS analytical procedures is accessible from Clementine.

For details on specific SPSS procedures, consult the documentation that came with your copy of SPSS. You can also click the SPSS Syntax Help button, available from the dialog box. This will provide syntax charts for the command that you are currently typing.

If you have trouble running SPSS Procedure nodes, consider the following tips:

- If field names used in Clementine are longer than eight characters (for versions prior to SPSS 12) or contain invalid characters, it is necessary to rename or truncate before reading into SPSS. For more information, see “Renaming or Filtering Fields for Export” on page 542.

- If SPSS was installed after Clementine, you may need to specify the SPSS installation directory in the Clementine Helper Applications dialog box.
- If SPSS windows are not opening properly, check options in SPSS to be sure that the program names and locations are set properly.

- Graphs generated with the Clementine option will still be displayed using the SPSS Viewer; however, the title of the graph itself will appear on the Outputs tab in the managers window.

**SPSS Procedure Node Syntax Tab**

Use this dialog box to create syntax code for SPSS procedures. Syntax is composed of two parts: a **statement** and associated **options**. The statement specifies the analysis or operation to be performed and the fields to be used. The options specify everything else, including which statistics to display, derived fields to save, and so on.

- If you have previously created syntax files, you can use them here by selecting Open from the File menu. Selecting an .sps file will paste the contents into the Procedure node dialog box.

- To insert previously saved syntax without replacing the current contents, select Insert from the File menu. This will paste the contents of an .sps file at the point specified by the cursor.

- If you are unfamiliar with SPSS syntax, the simplest way to create syntax code in Clementine is to first run the command in SPSS, copy the syntax into the SPSS Procedure node in Clementine, and execute the stream.

- Once you have created syntax for a frequently used procedure, you can save the syntax by selecting Save or Save As from the File menu.
Options for viewing and saving the output from the SPSS Procedure node are available below the syntax edit box:

**Figure 14-43**
*SPSS Procedure node dialog box*

- **Store.** Select the types of files you want to save from the SPSS Procedure node. You can save the following types of SPSS file formats:
  - **Data.** The data read into the node will be saved as a `.sav` file in the directory specified.
  - **SPSS Syntax.** The syntax used to run the SPSS procedure can be saved for later re-use. Syntax files are saved with an `.sps` extension.
  - **SPSS Results.** The results created with the SPSS Procedure node can be saved as a `.spo` file in the directory specified.

- **Path.** Specify a location for the output files selected.

- **Output Mode.** Select SPSS to display the results in an SPSS window (launched from the SPSS application installed on your machine). Select Clementine to use the default Clementine browser window. Only basic reports (frequencies, crosstabs, etc.) can be displayed in Clementine output mode.
Note: Graphs generated with the Clementine option will be displayed using the SPSS Viewer; however, the title of the graph itself will appear on the Outputs tab in the managers window.

**SPSS Procedure Output Browser**

Figure 14-44
SPSS Procedure output browser

<table>
<thead>
<tr>
<th>Frequencies</th>
<th>REGION</th>
<th>MAINCROP</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>Valid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Missing</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Frequency Table**

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Frequency</th>
<th>Percent</th>
<th>Valid Percent</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>midlands</td>
<td>93</td>
<td>31.0</td>
<td>31.0</td>
<td>31.0</td>
</tr>
<tr>
<td>north</td>
<td>39</td>
<td>13.0</td>
<td>13.0</td>
<td>44.0</td>
</tr>
<tr>
<td>southeast</td>
<td>125</td>
<td>41.7</td>
<td>41.7</td>
<td>85.7</td>
</tr>
<tr>
<td>southwest</td>
<td>43</td>
<td>14.3</td>
<td>14.3</td>
<td>100.0</td>
</tr>
<tr>
<td>Total</td>
<td>300</td>
<td>100.0</td>
<td>100.0</td>
<td></td>
</tr>
</tbody>
</table>

| Valid     |           |         |               |                    |
| maize     | 91        | 30.3    | 30.3          | 30.3               |
| potatoes  | 41        | 13.7    | 13.7          | 44.0               |
| rapeseed  | 49        | 16.3    | 16.3          | 60.3               |
| wheat     | 119       | 39.7    | 39.7          | 100.0              |
| Total     | 300       | 100.0   | 100.0         |                    |
The SPSS output browser shows you the results of the SPSS procedure that you executed if you selected Clementine output mode in the SPSS Procedure node. The usual saving, exporting, and printing options are available from the File menu, and the usual editing options are available from the Edit menu. For more information, see “Output Browser Menus” on page 489.

**Helper Applications**

To configure Clementine to work with SPSS and other applications, choose Helper Applications from the Tools menu.

**Figure 14-45**

*Helper Applications dialog box*

If SPSS is installed on your computer, you can configure Clementine to send data to SPSS using the SPSS Export node or the SPSS Procedure node.

**SPSS Interactive.** Enter the name of the command to execute SPSS in interactive mode (usually, *spsswin.exe* in the SPSS program directory).

**SPSS Production.** Enter the name of the command to execute SPSS in batch mode (usually, *spssprod.exe* in the SPSS program directory).

**Connection.** If SPSS Server is located on the same server as Clementine Server, you can enable a connection between the two applications, which increases efficiency by leaving data on the server during analysis. Select *Server* to enable the Port option below. The default setting is *Local*. 
**Port.** Specify the server port for SPSS Server.

**Other Helper Applications**

On the Other tab of the Helper Applications dialog box, you can specify the location of applications, such as AnswerTree and Excel, to work interactively with data from Clementine.

*Figure 14-46*
*
*Helper Applications dialog box: Other tab*

- **AnswerTree Interactive.** Enter the name of the command to execute AnswerTree (normally `atree.exe` in the AnswerTree program directory).
- **Excel(tm) Interactive.** Enter the name of the command to execute Excel (normally `excel.exe` in the Excel program directory).
- **Publish to Web URL.** Enter the URL for your SPSS Web Deployment Framework (SWDF) server for the Publish to Web option.
SuperNodes

SuperNode Overview

One of the reasons that Clementine's visual programming interface is so easy to learn is that each node has a clearly defined function. This means, though, that for complex processing, a long sequence of nodes may be necessary. Eventually, this may clutter the stream canvas and make it difficult to follow stream diagrams. There are two ways to avoid the clutter of a long and complex stream:

- You can split a processing sequence into several streams that feed one into the other. The first stream, for example, creates a data file that the second uses as input. The second creates a file that the third uses as input, and so on. You can manage these multiple streams by saving them in a project. A project provides organization for multiple streams and their output. However, a project file contains only a reference to the objects it contains, and you will still have multiple stream files to manage.

- A more streamlined alternative when working with complex stream processes is to create a SuperNode.

SuperNodes group multiple nodes into a single node by encapsulating sections of a data stream. This provides numerous benefits to the data miner:

- Streams are neater and more manageable.
- Nodes can be combined into a business-specific SuperNode.
- SuperNodes can be exported to libraries for reuse in multiple data mining projects.
Types of SuperNodes

SuperNodes are represented in the data stream by a star icon. The icon is shaded to represent the type of SuperNode and the direction in which the stream must flow to or from it.

There are three types of SuperNodes:

- Source SuperNodes
- Process SuperNodes
- Terminal SuperNodes

Figure 15-1
Types and shading of SuperNodes

Source SuperNodes

Source SuperNodes contain a data source just like a normal source node and can be used anywhere that a normal source node can be used. The left side of a source SuperNode is shaded to indicate that it is “closed” on the left and that data must flow downstream from a SuperNode.
Source SuperNodes have only one connection point on the right, showing that data leaves the SuperNode and flows to the stream.

**Process SuperNodes**

Process SuperNodes contain only process nodes and are unshaded to show that data can flow both *in* and *out* of this type of SuperNode.
Process SuperNodes have connection points on both the left and right, showing that data enters the SuperNode and leaves to flow back to the stream. Although SuperNodes can contain additional stream fragments and even extra streams, both connection points must flow through a single path connecting the From Stream and To Stream points.

*Note:* Process SuperNodes are also sometimes referred to as “Manipulation SuperNodes.”

**Terminal SuperNodes**

Terminal SuperNodes contain one or more terminal nodes (plot, table, etc.) and can be used in the same manner as a terminal node. A terminal SuperNode is shaded on the right side to indicate that it is “closed” on the right and that data can flow only into a terminal SuperNode.
Terminal SuperNodes have only one connection point on the left, showing that data enters the SuperNode from the stream and terminates inside the SuperNode.

Terminal SuperNodes can also contain scripts that are used to specify the order of execution for all terminal nodes inside the SuperNode. For more information, see “SuperNodes and Scripting” on page 572.

**Creating SuperNodes**

Creating a SuperNode “shrinks” the data stream by encapsulating several nodes into one node. Once you have created or loaded a stream on the canvas, there are several ways to create a SuperNode.

**Multiple Selection**

The simplest way to create a SuperNode is by selecting all of the nodes that you want to encapsulate:

- Use the mouse to select multiple nodes on the stream canvas. You can also use Shift-click to select a stream or section of a stream. *Note:* Nodes that you select...
must be from a continuous or forked stream. You cannot select nodes that are not adjacent or connected in some way.

Then, using one of the following three methods, encapsulate the selected nodes:

- Click the SuperNode icon (shaped like a star) on the toolbar.
- Right-click on the SuperNode, and from the context menu choose:
  Create SuperNode
  From Selection

- From the SuperNode menu, choose:
  Create SuperNode
  From Selection

![Figure 15-5](Creating a SuperNode using multiple selection)

All three of these options encapsulate the nodes into a SuperNode shaded to reflect its type—source, process, or terminal—based on its contents.

**Single Selection**

You can also create a SuperNode by selecting a single node and using menu options to determine the start and end of the SuperNode or encapsulating everything downstream of the selected node.

- Click the node that determines the start of encapsulation.
From the SuperNode menu, choose:
Create SuperNode
From Here

Figure 15-6
Creating a SuperNode using the context menu for single selection

SuperNodes can also be created more interactively by selecting the start and end of the stream section to encapsulate nodes:

- Click on the first or last node that you want to include in the SuperNode.
- From the SuperNode menu, choose:
  Create SuperNode
  Select...

  Alternatively, you can use the context menu options by right-clicking on the desired node.

  The cursor becomes a SuperNode icon, indicating that you must select another point in the stream. Move either upstream or downstream to the “other end” of the SuperNode fragment and click on a node. This action will replace all nodes in between with the SuperNode star icon.
Note: Nodes that you select must be from a continuous or forked stream. You cannot select nodes that are not adjacent or connected in some way.

**Nesting SuperNodes**

SuperNodes can be nested within other SuperNodes. The same rules for each type of SuperNode (source, process, and terminal) apply to nested SuperNodes. For example, a process SuperNode with nesting must have a continuous data flow through all nested SuperNodes in order for it to remain a process SuperNode. If one of the nested SuperNodes is terminal, then data would no longer flow through the hierarchy.

Figure 15-7
*Process SuperNode nested within another process SuperNode*

Terminal and source SuperNodes can contain other types of nested SuperNodes, but the same basic rules for creating SuperNodes apply.
Examples of Valid SuperNodes

Almost anything you create in Clementine can be encapsulated in a SuperNode. Following are examples of valid SuperNodes:

Figure 15-8
Valid process SuperNode with two connections in a valid stream flow

Figure 15-9
Valid terminal SuperNode including separate stream used to test generated models
Examples of Invalid SuperNodes

The most important aspect of creating valid SuperNodes is to ensure that data flows linearly through the SuperNode connections. If there are two connections (a process SuperNode), then data must flow in a stream from the beginning connector to the ending connector. Similarly, a source SuperNode must allow data to flow from the source node to the single connector that brings data back to the zoomed-out stream.
Figure 15-11
Invalid source SuperNode: Source node not connected to the data flow path

Figure 15-12
Invalid terminal SuperNode: Nested SuperNode not connected to the data flow path
Chapter 15

**Editing SuperNodes**

Once you have created a SuperNode, you can examine it more closely by zooming in to it. To view the contents of a SuperNode, you can use the zoom-in icon from the Clementine toolbar, or the following method:

- Right-click on a SuperNode.
- From the context menu, choose Zoom In.

The contents of the selected SuperNode will be displayed in a slightly different Clementine environment, with connectors showing the flow of data through the stream or stream fragment. At this level on the stream canvas, there are several tasks that you that can perform:

- Modify the SuperNode type—source, process, or terminal.
- Create parameters or edit the values of a parameter. Parameters are used in scripting and CLEM expressions.
- Specify caching options for the SuperNode and its subnodes.
- Create or modify a SuperNode script (terminal SuperNodes only).

**Modifying SuperNode Types**

In some circumstances, it is useful to alter the type of a SuperNode. This option is available only when you are zoomed in to a SuperNode, and it applies only to the SuperNode at that level. The three types of SuperNodes and their connectors are:

<table>
<thead>
<tr>
<th>SuperNode Type</th>
<th>Connectors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source SuperNode</td>
<td><img src="image" alt="Source Connector" /></td>
<td>One connection going out</td>
</tr>
<tr>
<td>Process SuperNode</td>
<td><img src="image" alt="Process Connector" /></td>
<td>Two connections: one coming in and one going out</td>
</tr>
<tr>
<td>Terminal SuperNode</td>
<td><img src="image" alt="Terminal Connector" /></td>
<td>One connection coming in</td>
</tr>
</tbody>
</table>

**To change the type of a SuperNode:**

- Be sure that you are zoomed in to the SuperNode.
Click the toolbar button for the type of SuperNode to which you want to convert.

Alternatively, you can use the SuperNode menu to choose a type. From the SuperNode menu choose SuperNode Type, and then choose the type.

**Annotating and Renaming SuperNodes**

You can rename a SuperNode as it appears in the stream as well as write annotations used in a project or report. To access these properties:

- Right-click on a SuperNode (zoomed out) and choose Rename and Annotate.

- Alternatively, from the SuperNode menu choose Rename and Annotate. This option is available in both zoomed-in and zoomed-out modes.

In both cases, a dialog box opens with the Annotations tab selected. Use the options here to customize the name displayed on the stream canvas and provide documentation regarding SuperNode operations.

*Figure 15-13*  
*Annotating a SuperNode*
SuperNode Parameters

In Clementine, you have the ability to set user-defined variables, such as Minvalue, whose values can be specified when used in scripting or CLEM expressions. These variables are called parameters. You can set parameters for streams, sessions, and SuperNodes. Any parameters set for a SuperNode are available when building CLEM expressions in that SuperNode or any nested nodes. Parameters set for nested SuperNodes are not available to their parent SuperNode.

There are two steps to creating and setting parameters for SuperNodes:

- Define parameters for the SuperNode.
- Then, specify the value for each parameter of the SuperNode.

These parameters can then be used in CLEM expressions for any encapsulated nodes.

Defining SuperNode Parameters

Parameters for a SuperNode can be defined in both zoomed-out and zoomed-in modes. The parameters defined apply to all encapsulated nodes. To define the parameters of a SuperNode, you first need to access the Parameters tab of the SuperNode dialog box. Use one of the following methods to open the dialog box:

- Double-click a SuperNode in the stream.
- From the SuperNode menu, choose Set Parameters.
- Alternatively, when zoomed in to a SuperNode, choose Set Parameters from the context menu.

Once you have opened the dialog box, the Parameters tab is visible with any previously defined parameters.

To define a new parameter:

- Click the Define Parameters button to open the dialog box.
**Name.** Enter the name of the parameter in the field. Do not include the $P-$ prefix that denotes a parameter in CLEM expressions. For example, to create a parameter for the minimum temperature, you could type minvalue.

**Long name.** Enter a long name, such as Minimum value, for the parameter. This is the user-friendly name that will be used in the SuperNode dialog box for specifying parameters.

**Storage.** Select a storage type from the drop-down list. Storage indicates how the data values are stored in the parameter. For example, when working with values containing leading zeros that you want to preserve (such as 008), you should select String as the storage type. Otherwise these zeros will be stripped from the value. Available storage types are String, Integer, Real, Time, Date, and Timestamp.

**Value.** Set a default value for when the SuperNode parameters have not been specified.

**Type (Optional).** If you plan to deploy the stream to an external application, select a usage type from the drop-down list. Otherwise, it is advisable to leave the type column as is.
Click the arrows at the right to create new parameters and move a selected parameter further up or down the list of available parameters. Use the delete button (marked with an X) to remove the selected parameter. These parameters are now listed on the Parameters tab of the SuperNode properties dialog box.

**Setting Values for SuperNode Parameters**

Once you have defined parameters for a SuperNode, you can specify values using the parameters in a CLEM expression or script.

**To specify the parameters of a SuperNode:**

- Double-click on the SuperNode icon to open the SuperNode dialog box.
- Alternatively, from the SuperNode menu choose Set Parameters.
- Click the Parameters tab. *Note:* The fields in this dialog box are the fields defined by clicking the Define Parameters button on this tab.
- Enter a value in the text box for each parameter that you have created. For example, you can set the value `minvalue` to a particular threshold of interest. This parameter can then be used in numerous operations, such as selecting records above or below this threshold for further exploration.

![Figure 15-15](image)

*Specifying parameters for a SuperNode*
**Using SuperNode Parameters to Access Node Properties**

SuperNode parameters can also be used to define node properties (also known as slot parameters) for encapsulated nodes. For example, suppose you want to specify that a SuperNode train an encapsulated Neural Net node for a certain length of time using a random sample of the data available. Using parameters, you can specify values for the length of time and percentage sample.

The example SuperNode contains a Sample node called “Sample” and a Neural Net node called “Train.” You can use the node dialog boxes to specify the Sample node's **Sample** setting as Random % and the Neural Net node's **Stop on** setting to Time. Once these options are specified, you can access the node properties with parameters and specify specific values for the SuperNode. In the SuperNode dialog box, click Define Parameters and create the following parameters:
Note: The parameter names, such as `Sample.rand_pct`, use correct syntax for referring to node properties, where `Sample` represents the name of the node and `rand_pct` is a node property. For more information, see “Properties Reference Overview” in Appendix D on page 737.

Once you have defined these parameters, you can easily modify values for the two Sample and Neural Net node properties without reopening each dialog box. Instead, simply select Set Parameters from the SuperNode menu to access the Parameters tab of the SuperNode dialog box, where you can specify new values for Random % and Time. This is particularly useful when exploring the data during numerous iterations of model building.

**Figure 15-18**
Specifying values for node properties on the Parameters tab in the SuperNode dialog box

---

**SuperNodes and Caching**

From within a SuperNode, all nodes except terminal nodes can be cached. Caching is controlled by right-clicking on a node and choosing one of several options from the Cache context menu. This menu option is available both from outside a SuperNode and for the nodes encapsulated within a SuperNode.
There are several guidelines for SuperNode caches:

- If any of the nodes encapsulated in a SuperNode have caching enabled, the SuperNode will also.
- Disabling the cache on a SuperNode disables the cache for all encapsulated nodes.
- Enabling caching on a SuperNode actually enables the cache on the last cacheable subnode. In other words, if the last subnode is a Select node, the cache will be enabled for that Select node. If the last subnode is a terminal node (which does not allow caching), the next node upstream that supports caching will be enabled.
- Once you have set caches for the subnodes of a SuperNode, any activities upstream from the cached node, such as adding or editing nodes, will flush the caches.
You can use the Clementine scripting language to write simple programs that manipulate and execute the contents of a terminal SuperNode. For instance, you might want to specify the order of execution for a complex stream. As an example, if a SuperNode contains a Set Globals node that needs to be executed before a Plot node, you can create a script that executes the Set Globals node first. Values calculated by this node, such as the average or standard deviation, can then be used when the Plot node is executed.

The Script tab of the SuperNode dialog box is available only for terminal SuperNodes.

To open the scripting dialog box for a terminal SuperNode:

- Right-click on the SuperNode canvas and choose SuperNode Script.

- Alternatively, in both zoomed-in and zoomed-out modes, you can choose SuperNode Script from the SuperNode menu.

*Note:* SuperNode scripts are executed only with the stream and SuperNode when you have selected Run this script in the dialog box.
Specific options for scripting and its use within Clementine are discussed elsewhere in this guide. For more information, see “Introduction to Scripting” in Chapter 18 on page 597.

**Saving and Loading SuperNodes**

One of the advantages of SuperNodes is that they can be saved and reused in other streams. When saving and loading SuperNodes, note that they use an `.slb` extension.

**To save a SuperNode:**

- Zoom in on the SuperNode.
- From the SuperNode menu, choose Save SuperNode.
- Specify a filename and directory in the dialog box.
Select whether to add the saved SuperNode to the current project.

Click Save.

**To load a SuperNode:**

- From the Insert menu in the Clementine window, choose SuperNode.
- Select a SuperNode file (.slb) from the current directory or browse to a different one.
- Click Load.

*Note:* Imported SuperNodes have the default values for all of their parameters. To change the parameters, double-click on a SuperNode on the stream canvas.
Chapter 16

Projects and Reports

Introduction to Projects

A **project** is a group of files related to a data mining task. Projects include data streams, graphs, generated models, reports, and anything else that you have created in Clementine. At first glance, it may seem that Clementine projects are simply a way to organize output, but they are actually capable of much more. Using projects, you can:

- Annotate each object in the project file.
- Use the CRISP-DM methodology to guide your data mining efforts. Projects also contain a CRISP-DM Help system that provides details and real-world examples on data mining with CRISP-DM.
- Add non-Clementine objects to the project, such as a PowerPoint slide show used to present your data mining goals or white papers on the algorithms that you plan to use.
- Produce both comprehensive and simple update reports based on your annotations. These reports can be generated in HTML for easy publishing on your organization’s intranet.

The projects tool is visible by default, but it can also be accessed by selecting Project from the View menu. Objects that you add to a project can be viewed in two ways: **Classes view** and **CRISP-DM view**. Anything that you add to a project is added to both views, and you can toggle between views to create the organization that works best.
By supporting the Cross-Industry Standard Process for Data Mining (CRISP-DM), Clementine projects provide an industry-proven and nonproprietary way of organizing the pieces of your data mining efforts. CRISP-DM uses six phases to describe the process from start (gathering business requirements) to finish (deploying your results). Even though some phases do not typically involve work in Clementine, the projects tool includes all six phases so that you have a central location for storing and tracking all materials associated with the project. For example, the Business Understanding phase typically involves gathering requirements and meeting with colleagues to determine goals rather than working with data in Clementine. The projects tool allows you to store your notes from such meetings in the Business Understanding folder for future reference and inclusion in reports.
The CRISP-DM projects tool is also equipped with its own Help system to guide you through the data mining life cycle. To access the CRISP-DM Help system from the Help menu, choose CRISP-DM Help.

**Setting the Default Project Phase**

Objects added to a project are added to a default phase of CRISP-DM. This means that you need to organize objects manually according to the data mining phase in which you used them. It is wise to set the default folder to the phase in which you are currently working.

*To select which phase to use as your default:*

- In CRISP-DM view, right-click on the folder for the phase to set as the default.
- From the menu, select Set as Default.

The default folder is displayed in bold type.
Classes View

The Classes view in the projects tool organizes your work in Clementine categorically by the type of objects created. Saved objects can be added to any of the following categories:

- Streams
- Nodes
- Models
- Tables, graphs, reports
- Other (non-Clementine files, such as slide shows or white papers relevant to your data mining work)

Figure 16-3
Classes view in the projects tool

Adding objects to the Classes view also adds them to the default phase folder in the CRISP-DM view.
Building a Project

A project is essentially a file containing references to all of the files that you associate with the project. This means that project items are saved both individually and as a reference in the project file (.cpj). Because of this referential structure, note the following:

- Project items must first be saved individually before being added to a project. If an item is unsaved, you will be prompted to save it before adding it to the current project.
- Objects that are updated individually, such as streams, are also updated in the project file.
- Manually moving or deleting objects (such as streams and nodes) from the file system will render links in the project file invalid.

Creating a New Project

New projects are easy to create in the Clementine window. You can either start building one, if none is already open, or you can close an existing project and start from scratch.

▶ From the stream canvas menus, choose:
  File
  Project
  New Project...

Adding to a Project

Once you have created or opened a project, you can add objects, such as data streams, nodes, and reports, using several methods.

Adding Objects from the Managers

Using the managers in the upper right corner of the Clementine window, you can add streams or output.

▶ Select an object, such as a table or a stream, from one of the managers tabs.
Right-click and select Add to Project. If the object has been previously saved, it will automatically be added to the appropriate objects folder (in Classes view) or to the default phase folder (in CRISP-DM view).

*Note:* You may be asked to save the object first. When saving, be sure that Add file to project is selected in the Save dialog box. This will automatically add the object to the project after you save it.

**Figure 16-4**
*Adding items to a project*

---

**Adding Nodes from the Canvas**

You can add individual nodes from the stream canvas using the Save dialog box.

- Select a node on the canvas.
Right-click and select Save Node. Alternatively, from the menus, you can choose:
- Edit
  - Node
  - Save Node...

In the Save dialog box, select Add file to project.

Create a name for the node and click Save. This saves the file and adds it to the project.

Nodes are added to the Nodes folder in Classes view and to the default phase folder in CRISP-DM view.

**Adding External Files**

You can add a wide variety of non-Clementine objects to a project. This is useful when managing the entire data mining process within Clementine. For example, you can store links to data, notes, presentations, and graphics in a project. In CRISP-DM view, external files can be added to the folder of your choice. In Classes view, external files can be saved only to the Other folder.

**To add external files to a project:**

- Drag files from the desktop to the project.
  
  or

- Right-click on the target folder in CRISP-DM or Classes view.

- From the menu, select Add to Folder.

- Select a file from the dialog box and click Open.

- This will add a reference to the selected object inside Clementine projects.

**Setting Project Properties**

You can customize a project's contents and documentation using the project properties dialog box. To access project properties:

- Right-click an object or folder in the projects tool and select Project Properties.
Click the Project tab to specify basic project information.

**Figure 16-5**

*Setting project properties*

![Screenshot of project properties window]

- **Author.** The default author name is detected from user settings on your computer. Make any adjustments for this project.

- **Created.** The project's creation date is displayed in an uneditable field.

- **Summary.** Create a summary for your data mining project that will be displayed in the project report.

- **Contents.** This uneditable table contains a list of the type and number of objects referenced by the project file.

- **Update file references when loading project.** Select this option to update the project's references to its components. *Note:* The files added to a project are not saved in the project file itself. Rather, a reference to the files is stored in the project. This means that moving or deleting a reference to the files will remove that object from the project.
Annotating a Project

The projects tool provides a number of ways to annotate your data mining efforts. Project-level annotations are often used to track “big-picture” goals and decisions, while folder or node annotations provide additional detail. The Annotations tab provides enough space for you to document project-level details, such as the exclusion of data with irretrievable missing data or promising hypotheses formed during data exploration.

To annotate a project:

1. Click the Annotations tab.

Folder Properties and Annotations

Individual project folders (in both CRISP-DM and Classes view) can be annotated. In CRISP-DM view, this can be an extremely effective way to document your organization’s goals for each phase of data mining. For example, using the annotation
tool for the *Business Understanding* folder, you can include documentation such as “The business objective for this study is to reduce churn among high-value customers.” This text could then be automatically included in the project report by selecting the *Include in report* option.

**To annotate a folder:**

- Select a folder in the projects tool.
- Right-click the folder and select *Folder Properties*.

In CRISP-DM view, folders are annotated with a summary of the purpose of each phase as well as guidance on completing the relevant data mining tasks. You can remove or edit any of these annotations.

**Figure 16-7**  
*Project folder with CRISP-DM annotation*

- **Name.** Displays the name of the selected field.
- **Tooltip text.** Create custom tooltips that will be displayed when you hover the mouse over a project folder. This is useful in CRISP-DM view, for example, to provide a quick overview of each phase's goals or to mark the status of a phase, such as “In progress” or “Complete.”
- **Annotation field.** Use this field for more lengthy annotations that can be collated in the project report. The CRISP-DM view includes a description of each data mining phase in the annotation, but you should feel free to customize this for your own project.
- **Include in report.** To include the annotation in reports, select *Include in report*. 
Object Properties

You can view object properties and choose whether to include individual objects in the project report. To access object properties:

▶ Right-click an object in the project window.
▶ From the menu, choose Object Properties.

Figure 16-8
Object properties dialog box

Name. Lists the name of the saved object.
Path. Lists the location of the saved object.
Include in report. Select to include the object details in a generated report.

Closing a Project

When you exit Clementine or open a new project, the existing project is closed, including all associated files. Alternatively, you can choose to close the project file itself and leave all associated files open. To close a project file:

▶ From the File menu, choose Close Project.
▶ You may be prompted to close or leave open all files associated with the project. Select Leave Open to close the project file (.cpj) itself but to leave open all associated files, such as streams, nodes, or graphs.

If you modify and save any associated files after the close of a project, these updated versions will be included in the project the next time you open it. To prevent this behavior, remove the file from the project or save it under a different filename.
Building a Report

One of the most useful features of projects is the ability to easily generate reports based on the project items and annotations. Reports can be generated immediately and viewed in the project properties dialog box, where they can also be printed and saved as HTML for distribution or display on your organization's Web site.

Before generating a report, you can select objects for inclusion as well as report properties by clicking the Generate Report button on the Report tab in the project properties dialog box.

Figure 16-9
Report tab with a generated report
Generating Reports

Reports are often generated from project files several times during the data mining process for distribution to those involved in the project. The report culls information about the objects referenced from the project file as well as any annotations created. You can create reports based on either the Classes or CRISP-DM view.

To generate a report:

- In the project properties dialog box, click the Report tab.
- Click the Generate Report button. This will open the report dialog box.

Figure 16-10
Selecting options for a report

The options in the report dialog box provide multiple ways to generate the type of report you need:

Report structure. Select either CRISP-DM or Classes view from the drop-down list. CRISP-DM view provides a status report with “big-picture” synopses as well as details about each phase of data mining. Classes is an object-based view that is more appropriate for internal tracking of data and streams.
Author. The default user name is displayed, but you can easily make changes for inclusion in the report.

Report includes. Use the radio buttons to select a method for including objects in the report. Select all folders and objects to include all items added to the project file. You can also include items based on whether Include in Report is selected in the object properties. Alternatively, to check on unreported items, you can choose to include only items marked for exclusion (where Include in Report is deselected).

Select. This option allows you to easily provide project updates by selecting only recent items in the report. Alternatively, you can track older and perhaps unresolved issues by setting parameters for old items. Select all items to dismiss time as a parameter for the report.

Order by. Using the drop-down list, you can select a combination of the following object characteristics to order them within a folder or phase:

- **Type.** Group objects by type within a phase.
- **Name.** Organize objects using alphabetical order.
- **Added date.** Sort objects using the date added to the project.

Save modified files before reporting. Select to save any objects within the project that have been modified.

**Saving and Exporting Reports**

The generated report is displayed as HTML in the project properties dialog box. You can save and print the report using the controls on the Annotations tab.
Figure 16-11
Generated report window

Use the buttons at the top of the HTML window to:

- Print the report using the Print button.
- Save the report as HTML by clicking the Save button to specify a report name and location. The HTML file can then be exported for use on the Web.
Data Understanding

The data understanding phase involves collecting data in order to become familiar with the data, identify data quality problems and identify initial insights or interesting subsets.

<table>
<thead>
<tr>
<th>Name</th>
<th>File Name</th>
<th>Added</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>goods.str</td>
<td>C:\Program Files\Clementine7.0\demo\goods.str</td>
<td>Jun 7, 2002 6:51:22 PM</td>
<td></td>
</tr>
<tr>
<td>goodplot.str</td>
<td>C:\Program Files\Clementine7.0\demo\goodplot.str</td>
<td>Jun 7, 2002 6:51:40 PM</td>
<td></td>
</tr>
</tbody>
</table>

Data Preparation

This phase covers all activities involved in constructing the final datasets to be used in the modeling phase.

<table>
<thead>
<tr>
<th>Name</th>
<th>File Name</th>
<th>Added</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>mmodels.pdf</td>
<td>C:\Documents and Settings\defoster\Desktop\mmodels.pdf</td>
<td>Jun 7, 2002 7:45:14 PM</td>
<td></td>
</tr>
</tbody>
</table>

Modeling

In this phase, various modeling techniques are selected and applied, and their parameters optimised.

<table>
<thead>
<tr>
<th>Name</th>
<th>File Name</th>
<th>Added</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>goodlearn.str</td>
<td>C:\Program Files\Clementine7.0\demo\goodlearn.str</td>
<td>Jun 7, 2002 6:52:00 PM</td>
<td></td>
</tr>
</tbody>
</table>

Evaluation

Models built by the modeling phase are evaluated and the steps involved in building them reviewed to ensure the business objectives have been achieved.
Batch Mode Execution

Introduction to Batch Mode

Data mining is usually an interactive process—you interact with data and with models to improve your understanding of the data and the domain it represents. However, Clementine streams can also be used to process data and perform data mining tasks in a batch, or non-interactive, manner by running Clementine with no visible user interface. This facility is called batch mode.

To perform a long-running or repetitive task without your intervention and without the presence of the user interface on the screen, use batch mode. Examples of such tasks include:

- Running a time-consuming modeling exercise in the background.
- Running a stream at a scheduled time (for example, overnight, when the resultant load on the computer will not be inconvenient).
- Running a data preprocessing stream on a large volume of data (for example, in the background and/or overnight).
- Regularly scheduled tasks, such as producing monthly reports.
- Running a stream as an embedded part of another process, such as a scoring engine facility.

Note: Clementine operations can be scheduled in batch mode using the appropriate operating system commands or utilities (for example, the `at` command under Windows NT).
Chapter 17

**Working in Batch Mode**

Working in batch mode typically involves:

- Invoking Clementine in batch mode using the `clemb` command.
- Connecting to a server or running in local mode.
- Loading an existing stream or script file.
- Executing the stream or script.

Once execution is complete, you can then consult the log file produced by default in batch mode and view the results of graphs, output nodes, and models. For more information on these steps, see the following topics.

**Invoking the Software**

Using the command line of your operating system, you can launch Clementine in batch mode. From both client and server machines, you can launch Clementine in batch mode using the following steps:

- Open a DOS window or command prompt window.
- Type the command `clemb` as well as any arguments (flags) used to load streams, execute scripts, and connect to a server.

**Using Command Line Arguments**

In order for Clementine to open and execute files, such as streams and scripts, in batch mode, you need to alter the initial command (`clemb`) that launches the software. There are a number of command line arguments, also referred to as flags, that you can use to:

- Connect to a server.
- Load streams, scripts, states, projects, and output files.
- Specify log file options.
- Set default directories for use in Clementine.
All of the above operations require the use of flags appended to the `clemb` command. Flags follow the form `-flag`, where the dash precedes the argument itself. For example, using the flag `-server` in conjunction with the initial argument `clemb` will connect to the server specified using other flag options.

You can combine the `clemb` command with a number of other startup flags, such as `-stream` and `-execute`, in order to load and execute streams in batch mode. The following command loads and executes the stream `report.str` without invoking the user interface:

```
clemb -stream report.str -execute
```

A complete list of command line arguments can be found in Appendix C.

- Clementine states and scripts are also executed in this manner, using the `-state` and `-script` flags, respectively. Multiple states and streams can be loaded by specifying the relevant flag for each item.

- Multiple arguments can be combined into a single command file and specified at startup using the `@` symbol. For more information, see “Combining Multiple Arguments” in Appendix C on page 732.

**Batch Mode Log Files**

Running in batch mode produces a log file. By default, the name of this log file is `clem_batch.log`, but you can specify an alternative name using the `-log` flag. For example, the following command executes `report.str` in batch mode and sends the logging information to `report.log`:

```
clemb -stream report.str -execute -log report.log
```

Normally, the log file overwrites any existing file of the same name, but you can make Clementine append to the log file instead by using the `-appendlog` flag. Logging can also be suppressed altogether by using the `-nolog` flag. For more information, see “Log File Arguments” in Appendix C on page 735.

*Note:* Logging arguments are available only when running in batch mode.
Chapter 17

**Scripting in Batch Mode**

In its simplest form, batch mode execution of Clementine streams is performed one at a time using the command line arguments discussed earlier in this guide. A given stream is executed without significantly altering its node parameters. While this may work well for automated production of monthly churn reports or predictions, it cannot handle the sophisticated processes that many advanced data miners would like to automate.

For example, a financial institution may want to construct a number of models using different data or modeling parameters, test the models on another set of data, and produce a report on the results. Because this process requires repetitive modifications to a stream and the creation and deletion of nodes, automating it requires the use of scripting. Scripting allows complex processes that would otherwise require user intervention to be automated and executed in batch mode. For more information, see “Introduction to Scripting” in Chapter 18 on page 597.

**To execute a script in batch mode:**

- Append the clemb command with the -script flag, specifying the name of the script you want to execute.

- Also use the -execute flag with the above arguments to execute the specified script. This will run the stand-alone script in its entirety.

For example, to load and execute a script that runs a model producing churn scores that are stored as output for the data warehouse, you would use the following command:

```
clemb -script clemscript.txt -execute
```

**Using Parameters in Batch Mode**

You can modify the effect of executing a stream in batch mode by supplying parameters to the command line launch of Clementine. These might be **simple parameters**, used directly in CLEM expressions, or they might be node properties, also called **slot parameters**, which are used to modify the settings of nodes in the stream.
For example, the following stream selects a subset of data from a file, passes it through a neural net, and sends the results to a file:

**Figure 17-1**  
*Stream operations in the user interface*

![Diagram of stream operations](image)

The value of the field *Month* determines the selected data; the expression in the Select node is:

\[ \text{Month} == \text{'$P-mth'} \]

When running the same stream in batch mode, select the appropriate month by setting the value of the parameter *mth* in the command line:

```
clemb -stream predict.str -Pmth=Jan -execute
```

*Note:* In command line arguments, the `-P` flag is used to denote a parameter.

Sometimes the required command line control of the stream involves modifying the settings of the nodes in the stream using slot parameters. Consider the following stream, which reads a file, processes its contents, and sends a report to another file:

**Figure 17-2**  
*Stream operations in the user interface*

![Diagram of stream operations](image)
Suppose that we want to generate the report once a month, reading the appropriate month's data and sending the report to a file whose name indicates the relevant month. We might want to set the filenames for the source data and for the report. The following command sets the appropriate slot parameters and executes the stream:

```
iclem -stream report.str -Porder.full_filename=APR_orders.dat -Preport.filename=APR_report.txt -execute
```

*Note:* This command does not contain the operating-system-specific code that schedules it to run monthly.

**Working with Output in Batch Mode**

Working with visual output, such as tables, graphs, and charts, typically requires a user interface. Since batch mode does not launch the Clementine user interface, output objects are diverted to a file so that you can view them later, either in the user interface or in another software package. Using the properties available for nodes, also called *slot parameters*, you can control the format and filename of output objects created during batch mode.
Scripting is a powerful tool used to automate tedious processes in the user interface and work with objects in batch mode. Scripts can perform the same kinds of actions that you perform with a mouse or a keyboard. You can set options for nodes and perform derivations using a subset of CLEM (Clementine Language for Expression Manipulation).

Typically, scripts automate tasks that would otherwise have to be performed by the user. These tasks might be highly repetitive or time consuming for the user to perform manually. Using scripts, you can:

- Gain control of the order of execution of a stream.
- Specify an automatic sequence of actions that normally involves user interaction—for example, you can build a model and then test it.
- Set up complex processes that require substantial user interaction—for example, cross-validation procedures that require repeated model generation and testing.
- Set up processes that manipulate streams—for example, you can take a model training stream, run it, and produce the corresponding model testing stream automatically.
- Automate Clementine processes from other applications or scheduling systems by invoking Clementine in batch mode to execute a script.

The following sections describe how to use scripts in greater detail.
Types of Scripts

There are three types of scripting used in Clementine:

- You can associate scripts with a particular stream and use them exclusively in conjunction with that stream. These scripts are saved and loaded with the stream to which they are attached and are called stream scripts. You can designate a stream script as the default method of using that stream. Thus, executing the stream means executing the script.

- Scripts can also be used in terminal SuperNodes to control the execution of the SuperNode contents. These are referred to as SuperNode scripts.

- Other scripts are not associated with any particular stream—for example, scripts that manipulate multiple streams cannot be associated with an individual stream. These scripts are stored in text files and are called standalone scripts.

Example Clementine Script

A stream can be used to train a neural network model when executed. Normally, to test the model, you would insert the model manually near the end of the stream, make the appropriate connections, and execute the Analysis node.

Using a Clementine script, you can automate the process of testing the model after you have created it. For example, you might use a script like this:

```plaintext
execute Drug:neuralnetnode
create analysisnode at 700 200
set DRUG1n:varfilenode.full_filename = "$CLEO_DEMOS\DRUG2n"
insert model Drug connected between :typenode and :analysisnode
execute :analysisnode
```

This script executes the Neural Net node called Drug and then creates an Analysis node and switches the data source to read a test data set, Drug2n. The generated model is inserted between the test data source and the Analysis node (with the appropriate connections), and the Analysis node is executed.
Stream scripts such as this can be saved with the stream and run whenever the stream is executed. This provides automation at the stream level for quicker model building.

**Scripting in the User Interface**

When you use Clementine interactively, scripts can be created and executed using the following dialog boxes:

- Stream Script
- SuperNode Script
- Standalone Script

**Using Scripts in Streams**

Scripts can be used to customize operations within a particular stream and saved with that stream. The most common use of stream scripts is to specify a particular execution order for the terminal nodes within a stream. The stream script dialog box is used to edit the script that is saved with the current stream.
To access the stream script dialog box:

► From the File or Tools menus, choose: Stream Properties

► Click the Script tab to work with scripts for the current stream.

Figure 18-2
Stream script dialog box

The toolbar icons in this dialog box enable you to perform the following operations:

- Import scripts from text files.
- Save scripts to text files.
- Execute the current script.
- Execute selected lines from a script.
- Check the syntax of the current script.
In addition, this dialog box allows you to specify whether the script dictates the stream's default execution method. You can select Run this script to run the script every time the stream is executed and use the execution order specified in the script. The default setting is to Ignore this script during stream execution unless specifically activated by executing from within this dialog box.

**Modifying Stream Execution**

When a stream is executed, its terminal nodes are executed in an order optimized for the default situation. In some cases, you may prefer a different execution order. To modify the execution order of a stream, complete the following steps:

- Begin with an empty script.
- Click the Append default script button on the toolbar to add default stream script.
- Change the order of statements in the default stream script to the order in which you want statements to be executed.

**Script Checking**

You can quickly check the syntax of all types of scripts by clicking the red check button on the toolbar of the scripting dialog box. Script checking alerts you to any errors in your code and makes recommendations for improvement. To view the line with errors, click on the feedback in the lower half of the dialog box. This highlights the error in red.

**Example Stream Script**

The following stream script is used to create an if then else expression that counts the number of system-missing values (also called nulls) per record. The # character is used to indicate comments describing the script.

```plaintext
# Moves current nodes
position DRUG1n at 50 50
position :filternode at 150 50
position :typenode at 250 50
```
The resulting stream includes the newly created Derive node and a Table node used to view the values of the new field, \textit{nblanks}.

\textbf{Figure 18-3}

\textit{Resulting stream}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{resulting_stream.png}
\end{figure}

\textbf{Using Scripts in SuperNodes}

You can use the Clementine scripting language to write simple programs that manipulate and execute the contents of a terminal SuperNode. SuperNodes with scripts are useful for creating template streams. In addition, scripts in SuperNodes allow you to have more than one script running within a stream.
For example, you might want to specify the order of execution for a complex stream. If a SuperNode contains a Set Globals node that needs to be executed before deriving a new field used in a Plot node, you can create a script that executes the Set Globals node first. Values calculated by this node, such as the average or standard deviation, can then be used when the Plot node is executed.

When using scripts in SuperNodes, it is important to keep the following guidelines in mind:

- Only terminal SuperNodes can execute scripts. Therefore, the Scripts tab of the SuperNode dialog box is available only for terminal SuperNodes.
- Nodes within SuperNodes can access SuperNode, stream, and global parameters. However, nodes outside SuperNodes can access only stream and global parameters.
- You can also specify node properties within a SuperNode.

**To open the scripting dialog box for a terminal SuperNode:**

- Right-click on the SuperNode canvas, and from the context menu choose SuperNode Script.
- Alternatively, in both zoomed-in and zoomed-out modes, you can choose SuperNode Script from the SuperNode menu.

For more information, see “SuperNodes and Scripting” in Chapter 15 on page 572.

**Example SuperNode Script**

The following SuperNode script is used to specify the order of execution for terminal nodes inside the SuperNode.

```plaintext
execute 'Set Globals'
execute 'gains'
execute 'profit'
execute 'age v. $CC-pep'
execute 'Table'
```
Reordering operations within the SuperNode allows access to the globals created using a Set Globals node.

**Using Standalone Scripts**

The Standalone Script dialog box is used to create or edit a text file containing a script. It displays the name of the file and provides facilities for loading, saving, importing, and executing scripts. To access the Standalone Script dialog box, from the Tools menu, choose Standalone Script.

Figure 18-4
Standalone Script dialog box

```plaintext
clear stream
load stream "CLEO_DEMOS\DRUGlearn.str"
execute :c50node
save model Drug as rule.gm
clear stream
clear generated
load stream "CLEO_DEMOS\DRUGplot.str"
load model rule.gm
disconnect :plotnode
insert model Drug connected between :derive and :plot
set :plotnode.color_field = 'f-C-Drug'
execute :plotnode
```

Warning: 'clear generated' has been replaced by 'clear generated palette' on line 6 column 7

The same toolbar and script-checking options are available for standalone scripts as for stream scripts. A useful feature for all types of scripts is the ability to view feedback on scripting errors in the feedback panel at the bottom of the dialog box.
**Example Standalone Script**

Standalone scripts are useful for stream manipulation. Suppose that you have two streams—one that creates a model and another that uses graphs to explore the generated ruleset from the first stream with existing data fields. A standalone script for this scenario might look something like this:

```plaintext
clear stream
load stream "$CLEO_DEMOS\DRUGlearn.str"
execute :c50node
save model Drug as rule.gm
clear stream
clear generated palette
load stream "$CLEO_DEMOS\DRUGplot.str"
load model rule.gm
disconnect :plotnode
insert model Drug connected between :derive and :plot
set :plotnode.color_field = '$C-Drug'
execute :plotnode
```

**Figure 18-5**
*Resulting stream*
Scripting in Batch Mode

Scripting enables you to run operations typically performed in the user interface. Simply specify and execute a standalone stream at the command line when launching Clementine in batch mode. For example:

clemb -script scores.txt -execute

The -script flag loads the specified script, while the -execute flag executes all commands in the script file. For more information, see “Working in Batch Mode” in Chapter 17 on page 592.
Clementine External Module Interface

Introduction to the Clementine External Module Interface

The Clementine External Module Interface (CEMI) is a mechanism that allows the addition of other programs—for example, data processing routines or modeling algorithms—to Clementine as new nodes.

To do this, Clementine requires details about the external program, such as what it is called, what command parameters should be passed to the program, how Clementine should present options to the program and results to the user, and so forth. A text file called a specification file provides this information. Clementine translates the information in this file into a new node definition.

How CEMI Works

The CEMI specification file is a text file containing structured specifications describing the behavior of the new node. When it is executed, it launches an external application. The CEMI specification file describes what kind of data will be read into the node from Clementine and what sort of data will be received back from the node once execution is complete.

One of the most important characteristics to define is the type of node to create, which is determined largely by the sort of application that it accesses.

- **Source** nodes generate new data for Clementine.
- **Process** nodes take data from Clementine, modify it, and return the modified data to Clementine.
- **Terminal** nodes take data from Clementine and do something with it (display it, save it to a file, etc.).

- **Modeling** nodes take data from Clementine and create a model that can be browsed to learn something about the data.

After creating the CEMI specification file, load the new node into Clementine using the Palette Manager. This adds the new node to the appropriate palette and makes the node ready for use.

**System Architecture**

The use of external modules within the Clementine data mining system results in a default system architecture as shown below.

**Figure 19-1**
*External module system architecture*
When the stream encounters an external module, it writes the data to a temporary data file. The external module uses this data file as input along with any necessary data mapping. The external module then executes, typically generating an output data file that is read back into Clementine and passed to the next node.

This architecture forms the basis for external module processing in Clementine, and the CEMI specification file provides the guidelines for how this is done.

**Specification File**

Within a specification file, the main sections are introduced with their headings. The relevant subsections for the node type are then listed. At most, the file contains three sections.

The **NODE** section is the introductory section, or header, of the CEMI specification file and identifies the following:

- Name of the node
- Type of node
- Palette containing the node
- Custom icon used for node

The **CORE** section specifies the main body of the CEMI specification file and is constructed from subsections covering the following areas:

- **Parameters.** Specifies parameters (similar to variables) used throughout a specification file.
- **Execution.** Identifies the location of the external executable program.
- **Options.** Lists arguments to be passed to the program.
- **Appearance.** Defines the controls for the node's editing dialog box.
- **Inputs.** Specifies the data and the data model that flow from Clementine to the external module.
- **Outputs.** Specifies the data and the data model that flow from the external module back to Clementine.
- **Metafiles.** Provides information about the fields used by the program.
Chapter 19

- **Results.** Specifies what happens to the final stream results.
- **Return codes.** Associates text messages with values returned by the program.

The MODEL section has a structure similar to the CORE section but defines how the node for a model that is generated from the CEMI node behaves. Custom nodes are specified using an ICON subsection identical to that used in the NODE section. Only modeling nodes require a MODEL section. For more information, see “Specification File Overview” in Appendix E on page 799.

**Parameters**

Parameters are the attributes associated with the new node. They store specific information about each node, which can be used to control the functioning of the node and can be passed to the external module. For example, a parameter can store the current mode of operation for an algorithm with three possible modes. You can also add a user interface to your node to allow users to set or select values of parameters. In addition, parameters can reference input and output files. Define parameters in the PARAMETERS subsection.

**External Program**

Define the location of the external program either by explicitly entering the full path to the file or by referencing a parameter in the node. If the user has control over where the executable program resides, the second approach may be helpful because it allows the user to define the location in the editing dialog box. The EXECUTE subsection defines the location of the executable program(s) associated with the external module.

**Command Line Options**

Options define which parameters are passed as command arguments to the external program and how they should be passed. Each option consists of an expression usually composed of a parameter name concatenated with a string. Options can be passed to the program either unconditionally or conditionally. Define the options on the command line in the OPTIONS subsection of the specification file.
**Editing Dialog Box**

The editing dialog box provides an interface that enables the user to modify execution settings. The appearance of this dialog box is very important; it is where the node behavior is altered and modified. The interface must contain all of the necessary information and also must be easy to use.

**Settings Tab**

By default, controls occur on the Settings tab of a CEMI editing dialog box. You can define the controls for the node's editing dialog box in the CONTROLS subsection of the specification file. The dialog box may include a variety of controls, including option buttons, check boxes, text boxes, and menus. The type of parameter modified by the control determines which control appears in the dialog box, with some types providing alternate controls. You may group options on new tabs using the TAB option in the CONTROLS subsection. For more information, see “Tabs” in Appendix E on page 818.

![Figure 19-2](image-url)

*Settings tab for a text-mining CEMI*
When defining the controls, consider the following guidelines:

- Use the correct label as a descriptor for the control. It should be reasonably concise while conveying the correct information.

- Use the right parameter for a control. For example, a parameter that takes only two values does not necessarily require a check box. The Clementine C5.0 editing dialog box offers the option of selecting the output type as one of two values—Decision tree or Rule set. This setting could be represented as an option labeled Decision tree. When selected, the output type is decision tree; when deselected, the output is a ruleset. Although the outcome would be the same, using option buttons makes it easier for the user to understand.

- Controls for filenames are generally positioned at the top.

- Controls that form the focus of the node are positioned high in the dialog box. For example, graph nodes display fields from the data. Selecting those fields is the main function of the editing dialog box, so field parameters are placed at the top.

- Check boxes or option buttons often allow the user to select an option that needs further information. For example, selecting Use boosting in the C5.0 editing dialog box requires that the analysis include a number indicating Number of trials. The extra information is always placed after the option selection, either at the right side or directly beneath it.

The editing dialog boxes produced for the CEMI use Clementine's commit editing; the values displayed in the dialog boxes are not copied to the node until the user clicks OK, Apply, or in the case of terminal nodes, Execute. Similarly, the information displayed by the dialog box is not updated (for example, when the input fields to the node have changed as a result of operating upstream of the current node) until the user cancels and redispies the dialog box or clicks the Refresh button.

**Specification Tab**

The CEMI specification that was used to create a CEMI node is stored as part of the node itself, visible on the Specification tab of the editing dialog box.
This enables you to open streams containing CEMI nodes without having the CEMI node loaded in your version of Clementine. You cannot run the node, however. *Note:* Streams created prior to release 7.0 require that you load the CEMI specification beforehand, however, since previous versions didn’t save the specification as part of the node.

Once you have loaded a CEMI node into a stream, note that there is no way to change the specification of that node. Instead, replacing the specification of a loaded CEMI will only replace the specification used to create new nodes. Existing nodes will use the specification they were created with.

**Input and Output Fields**

The *data model* represents the structure of the data flowing through the stream. Describing the data at that point in the stream, the model corresponds to the information in the Type node. It lists the names of existing fields (not including those that have been filtered out) and describes their type.
Chapter 19

When adding any node to Clementine, consider how the data model passed into the node affects the behavior of that node. For example, a process node, such as a Derive node, takes an input data model, adds a new field to it, and produces an output data model that is passed to the next node in the Clementine stream. In contrast, a terminal node, such as a Graph node, takes an input data model and produces no output data model because the data is not passed to any more nodes. Clementine must know what will happen to the data model so that subsequent nodes can present the correct information about which fields are available. The data model information in the specification file gives Clementine the information necessary to keep the data model consistent across the entire stream.

Depending on whether data flows into, out of, or through the node, the specification file must describe the data model for input, output, or both. The external program can affect the data model either by adding new fields to whatever fields pass into the node or by replacing the fields coming into the node with new fields generated by the program itself. The INPUT_FIELDS and OUTPUT_FIELDS subsections of the specification file describe the effects of the CEMI node on the data model.

Input and Output Files

The data file created by Clementine and used as input to the external program, as well as any data file returned by the program, contain the data in a particular format. For example, it uses a specific character to separate data values, such as a comma. Furthermore, the file may or may not contain field names. The INPUT_DATA and OUTPUT_DATA subsections describe the format of these files, ensuring compatibility between Clementine and the external program with respect to data transfer.

Metafiles

Metafiles contain extra information about the data being passed to the program. This may be type information, or it may describe how each field will be transformed by the external program. Because metafiles can take a wide variety of formats, specifying them can be complex. The MAPPING_FILE subsection defines the structure of the metafile created by the CEMI node.
Results

In addition to output data files, the external program may generate results, such as a graphical display or a simple text report. The RESULTS subsection determines how to handle these results. The output can be viewed in text or HTML browsers or displayed using a specified external viewer.

Return Codes

Most programs perform some sort of error checking and display any necessary messages to the user. The programs typically return integers to indicate successful completion or other status. The RETURN_CODE subsection enables the handling of these integers by associating them with message text. The messages can be used to notify the user of an incorrect input value or a possible problem in the program.

Restrictions

When designing a CEMI node, some CEMI restrictions may influence how to structure the specification file:

- Generated model nodes produced by external programs cannot be exported.
- CEMI nodes that make any selections based on type require a Type node directly upstream. Such CEMI nodes automatically instantiate the data if a Type node is present.
- CEMI process nodes that extend the data model are more efficient if they include all fields than if they select a subset of the fields. Also, external modules that extend the data model must include the original input fields with their output fields.
- Restricting which fields are passed to the program applies only to terminal nodes. Process nodes and generated model nodes that extend the data model must be able to handle all of the fields that are passed into the node and return the original values in each record. This is in addition to handling any extra fields added by the node.
As an illustration of field passing, consider a generated model that classifies each record as a *YES* or a *NO* with a confidence value. The input file might be as follows:

```
M,Y,23,32562
M,N,34,13946
F,N,19,16231
....
```

In this case, the output file for the node must have the following form:

```
M,Y,23,32562,YES,0.78
M,N,34,13946,NO,0.46
F,N,19,16231,YES,0.61
....
```

The output file includes both the original values and the new fields.

**Example Specification File**

The following example creates a source node that runs the LexiMine application for text mining. This example highlights the general approach to creating a node, however, it does not address issues involved in the text-mining process. The section-by-section discussions of the specification file illustrate the issues involved in creating a CEMI node.

**Node Specification**

The name of the node is *LexiMine*, with the same text used as the label for the icon. This node will go on the source palette, and the data will not be hashed. These settings yield the following NODE specification:

```
NODE
    NAME LexiMine
    TITLE 'LexiMine'
    TYPE SOURCE
    PALETTE SOURCE
    HASH_DATA false
    ICON
        STANDARD 'c:/CEMI/images/standardLexi.gif'
        SMALL 'c:/CEMI/images/smallLexi.gif'
ENDICON
ENDNODE
```
Core Specification

Defining the core functionality for the node consists of the following:

- Defining parameters associated with the node.
- Supplying execution information for the program.
- Specifying options passed to the program.
- Designing the editing dialog box for the node.
- Defining the format of the output from the program.
- Handling the return codes from the program.
- Optionally, you can specify a custom node used for the module. For more information, see “Creating CEMI Node Icons” on page 627.

Parameters

This node uses several parameters of varying types:

```
PARAMETERS
    Outfile pathname 'Clemexport'
    db_build flag false
    data_dir text 'C:/LexiQuest/LexiMine/web'
    db_dir text 'C:/LexiQuest/LexiMine/db'
    lm_dir text 'C:/LexiQuest/LexiMine/sbin/leximine'
    db_name text ''
    read_doc_names flag false
    read_concepts flag true
    doc_type set oneof [0 1 2] 0
    text_unity set oneof [document paragraph] document
    para_sz_min number [10 50] 10
    para_sz_max number [300 3000] 300
    lang set oneof [1 2 3 4 5] 2
    freq number [1 3] 1
    filter_html flag true
    ext_table flag true
    ram number [1 500] 100
ENDPARAMETERS
```

The first eight parameters involve the database used for mining:

- The pathname parameter *Outfile* is a stub used for a generated database, which is *Clemexport* by default.
The text parameters \textit{data\_dir}, \textit{db\_dir}, and \textit{lm\_dir} all represent locations for files used during node execution. The parameter \textit{db\_name} corresponds to the name of the database and must be specified by the user.

The flag parameters determine how execution handles the specified database. The database will be rebuilt if \textit{db\_build} is true. The other two flag parameters, \textit{read\_doc\_names} and \textit{read\_concepts}, indicate whether or not document names and concepts should be read. By default, only concepts are read.

The next four parameters correspond to options for the document:

- The parameters \textit{doc\_type} and \textit{text\_unity} each define a set of values from which the user can select one entry.
- The number parameters \textit{para\_sz\_min} and \textit{para\_sz\_max} each equal a value within the defined numerical ranges.

The remaining parameters present general options:

- The parameter \textit{lang} defines a set of values representing five different languages for the analysis.
- The number parameters \textit{freq} and \textit{ram} define ranges for frequency and memory options.
- The flag parameters \textit{filter\_html} and \textit{ext\_table} control two general processing options, which are applied by default.

\textit{Execution}

After defining the parameters, designate the location of the program to run. The executable file is \texttt{c:/Program Files/CEMI/lclem\_wrap.exe}, yielding an \texttt{EXECUTE} subsection of:

\begin{verbatim}
EXECUTE
   COMMAND 'c:/Program Files/CEMI/lclem\_wrap.exe'
ENDEXECUTE
\end{verbatim}
Clementine External Module Interface

Options

The executable program accepts several options that define the analysis, defined in the OPTIONS subsection:

```
OPTIONS
   NOPARAM ['lexiclem.exe']
   db_build ['-new']
   NOPARAM ['-'] << text_unify
   NOPARAM ['-freq=' << 'freq' << '']
   NOPARAM ['-lang=' << lang]
   NOPARAM ['-struc=' << doc_type]
   NOPARAM ['-p9' << db_name]
   NOPARAM ['-r' << data_dir << '']
   filter_html ['-a']
   ext_table ['-t']
   NOPARAM ['-m' << 'ram' << '']
   NOPARAM ['END']
   NOPARAM ['-ldir ' << lm_dir]
   NOPARAM ['-gen ' << db_dir << 'Outfile']
   NOPARAM ['OUTfile']
   NOPARAM ['-out "c:/Program Files/CEMI/outfile.txt"']
ENDOPTIONS
```

The contents of the brackets are passed to the executable program as a single string of options, with options separated by spaces. Options beginning with NOPARAM are always passed. Option lines beginning with a flag parameter are passed only when the parameter has a value of true. The options themselves consist of so-called packet labels—indicators of the option being passed—which are often concatenated with parameter values using the “><” operator. The executable program uses the packet labels to process the options correctly. For example, the -p9 packet label identifies the option being passed as the database name.

Controls

The CONTROLS subsection determines the structure of the editing dialog box associated with the node. This dialog box allows the user to change parameters from their default settings.

```
CONTROLS
   SHEET
      NAME sheet_database
      TITLE 'Set Databases'
```
In this case, related controls are grouped into sheets to simplify finding the control on the dialog box. Each parameter that can be modified appears in the CONTROLS section with a label identifying what the control changes. Each parameter type has a default control in the dialog box, but some types have alternate controls. For example, the Set parameters are defined as menus. The values for Set parameters are assigned descriptive labels that are used in the dialog box in place of the values. By default, all controls are enabled, but the specification can override this behavior, making enablement depend on the value of parameter. For instance, the user can change the extraction limits only when selecting Paragraph Mode for Textual Unity.

Output Fields

The next step is to specify how data flows through the node. Because this is a source node, it takes no input and only the node output needs to be defined.
CREATE_IF [read_doc_names] NAME ['Document'] TYPE [AUTO]
CREATE NAME ['DocID'] TYPE [AUTO]
CREATE_IF [read_concepts] NAME ['Concept'] TYPE [AUTOSYMBOL]
CREATE_IF [read_concepts] NAME ['Data'] TYPE [AUTONUMBER]
CREATE_IF [read_concepts] NAME ['Type'] TYPE [AUTOSYMBOL]
ENDOUTPUT_FIELDS

The node always creates a field named DocID. The creation of other fields depends on the values of flag parameters. When reading document names, the node creates a field name Document. When reading concepts, three new fields are created. In addition to names for the fields, specify their types too.

**Output Data**

The format of the input and output data must be defined. Source nodes do not accept input, so only the latter is required.

OUTPUT_DATA
   FILE_NAME ['c:/Program Files/CEMI/outfile.txt']
   SEPARATOR ';'  
   INC_FIELDS false
ENDOUTPUT_DATA

The name of the file containing the data generated by the executable file is c:/Program Files/CEMI/outfile.txt, and it does not contain field names. Records in this file are separated by semicolons. This information allows the node to pass the generated data to the next node in the stream.

**Return Codes**

The program returns one of three values reporting on the status of the execution. The RETURN_CODE subsection assigns text to integers returned by the executable program.

RETURN_CODE
   SUCCESS_VALUE 0
   1 'Illegal option'
   2 'Must select either Read Document Names or Read Concepts'
ENDRETURN_CODE
Complete Specification File

The complete specification file is as follows:

SPECFILE
  NODE
    NAME LexiMine
    TITLE 'LexiMine'
    TYPE_SOURCE
    PALETTE_SOURCE
    HASH_DATA false
    ICON
      STANDARD '$CLEO\CEMI\images\lg_cemi_icon.gif'
      SMALL '$CLEO\CEMI\images\sm_cemi_icon.gif'
  ENDICON
ENDNODE

CORE
PARAMETERS
# Files
  Outfile pathname 'Clemexport'
# Database Sheet
  db_build flag false
  data_dir text 'C:/LexiQuest/LexiMine/web'
  db_dir text 'C:/LexiQuest/LexiMine/db'
  lm_dir text 'C:/LexiQuest/LexiMine/sbin/leximine'
  db_name text ''
  read_doc_names flag false
  read_concepts flag true
# Document Options
  doc_type set oneof [0 1 2] 0
  text_unity set oneof [document paragraph] document
  para_sz_min number [10 50] 10
  para_sz_max number [300 3000] 300
# Language Options
  lang set oneof [1 2 3 4 5] 2
  freq number [1 3] 1
# Other
  filter_html flag true
  ext_table flag true
  ram number [1 500] 100
ENDPARAMETERS

EXECUTE
  COMMAND 'c:/Program Files/CEMI/lclem_wrap.exe'
ENDEXECUTE

OPTIONS
  NOPARAM ['lexiclem.exe']
  db_build ['-new']
  NOPARAM ['-'] << text_unity
  NOPARAM ['-freq' '<< freq >> '']
Clementine External Module Interface

NOPARAM ['-lang=' << lang]
NOPARAM ['-struc=' << doc_type]
NOPARAM ['-p9' << db_name]
NOPARAM ['-r'' '<data_dir>' << ''']
filter_html ['-'a']
ext_table ['-'t']
NOPARAM ['-m' << '' ' ' 'ram ' << ''']
NOPARAM ['' ' ' 'END' ]
NOPARAM ['-ldir ' << lm_dir]
NOPARAM ['-gen ' '<db_dir>'<<'\\'>'<<db_name>'<<'\\'>'<<Outfile]
NOPARAM []' ' << read_doc_names ' << read_concepts ' << ''']
NOPARAM ['.txt']
NOPARAM ['-out "c:/Program Files/CEMI/outfile.txt"']
ENDOPTIONS

CONTROLS

SHEET
 NAME sheet_database
 TITLE 'Set Databases'
 db_build LABEL 'Always Rebuild Database'
 data_dir LABEL 'Specify Data Directory'
 db_name LABEL 'Set Database Name'
 read_doc_names LABEL 'Read Document Names'
 read_concepts LABEL 'Read Concepts'
 ENDSHEET

SHEET
 NAME sheet_document
 TITLE 'Document Options'
 doc_type LABEL 'Document Type' CONTROL MENU
 VALUES [[0 'Full Text'][1 'Structured Text'][
 [2 'XML Text'][
 text_unity LABEL 'Textual Unity' CONTROL MENU
 VALUES [[document 'Document Mode'][
 [paragraph 'Paragraph Mode'][
 ENABLED [doc_type 'Full Text']
 NOPARAM ''
 para_sz_min LABEL 'Min Extract Size for Paragraph'
 ENABLED [text_unity 'Paragraph Mode']
 para_sz_max LABEL 'Max Extract Size for Paragraph'
 ENABLED [text_unity 'Paragraph Mode']
 ENDSHEET

lang LABEL 'Language' CONTROL MENU
 VALUES [[1 'French'][2 'English'][3 'English-French'][
 [4 'German'][5 'Spanish']]
 freq LABEL 'Frequency'
 ENDCONTROLS

OUTPUT_FIELDS
 REPLACE
 CREATE_IF [read_doc_names] NAME ['Document'] TYPE [AUTO]
 CREATE NAME ['DocID'] TYPE [AUTO] 
 CREATE_IF [read_concepts] NAME ['Concept'] 
 TYPE [AUTOSYMBOL]
CEMI Node Management

After creating a specification file, load the new node into Clementine using the CEMI tab of the Palette Manager.

Figure 19-4
Palette Manager

The CEMI tab contains a selectable list of any previously loaded CEMI nodes along with Add and Remove buttons.
The Add button loads a new node (custom or default) corresponding to a specification file into the palette defined in the file. If the node loads successfully, the CEMI Specifications list updates to include the name of the added node. Clementine reports any errors occurring during loading with context information for the error, including the line number in the specification file where the problem arises. After loading, the CEMI node will be available to all users of the client.

The Remove button eliminates a CEMI node from the palette. Nodes of the removed type that appear in existing streams are not removed from the streams and will continue to function as before.

**Adding a CEMI Node**

Using a dialog box in Clementine, you can load specification files, one at a time, while the software is running.

- From the Tools menu of the stream canvas, choose CEMI.
- In the Palette Manager, click Add.
- In the Open dialog box, select the specification file for the node to be added. By default, the manager opens to the CEMI subdirectory of your Clementine installation.
- Click Open.

The new node appears on the palette designated in the specification file, and the name of the node appears in the list of CEMI specifications in the Palette Manager. Click OK to close the Palette Manager.

**Installing from the Command Line**

You can also add a CEMI node using the following command line argument:

-`-install_cemi <file>`

For example:

`clemb -install_cemi 'c:\cemi\test.spc'`
To install multiple specification files, repeat the argument.

- If an identical CEMI node exists, it will be replaced by the new one.
- When this argument is specified, all other options are ignored and Clementine will not be invoked.

**Removing a CEMI Node**

- From the Tools menu on the stream canvas, select CEMI.
- In the Palette Manager, select the node to be removed from the list of CEMI specifications.
- Click Remove.
- Click OK to close the Palette Manager.

**Uninstalling from the Command Line**

You can also remove a CEMI node using the following command line argument:

```
-uninstall_cemi <file>
```

For example:

```
clemb -uninstall_cemi 'c:\cem\test.spc'
```

- To remove multiple specification files, repeat the argument.
- When this argument is specified, all other options are ignored and Clementine will not be invoked.
Creating CEMI Node Icons

For each CEMI node and any associated generated models, you can create custom icons. There are two parts to creating and implementing custom icons:

■ Create icons conforming to Clementine requirements using a graphics package.
■ Specify the icons in the CEMI specification file. For more information, see “ICON Subsection” in Appendix E on page 803.

This topic provides guidelines for creating icons that display well in Clementine.

Graphics Requirements

Custom CEMI nodes should conform to the following standards:

■ CEMI nodes require both small and standard icons in order to conform with stream layout settings for icon size.
■ Standard-sized icons (used to display nodes on the stream canvas) are 48 pixels wide by 48 pixels high.
■ Small-sized icons (used to display nodes in the palette) are 36 pixels wide by 36 pixels high.
■ Color depth should be 16 colors or higher. Most application icons are 4-bit 16 color images.

Example CEMI icons are included in the cemi/icons/ directory of your Clementine installation.

Icon Layers

Node icons are composed of overlapping layers—the **glyph layer** and the **background layer**. You may work with them as a layered image or separately if your graphics package does not support layers.

**Glyph layer.** The glyph layer is the customizeable part of a node icon.
Chapter 19

Figure 19-5
*Glyph layer for a CEMI icon including transparent background*

- The central image represents the domain, or type of data mining performed (in this example, an open book for text mining).
- The CEMI image is a plug icon used to represent that the node is a CEMI. It is not required that you include this image in your custom icon.

The glyph layer should be mostly transparent, since it is “painted” on the background layer in Clementine. An aqua blue background has been used here to represent transparency.

**Background layer.** The background layer is the same for all nodes and includes both the node background color and border. It is not recommended that you alter the background layer, since it should be visible through the transparent glyph layer in order to provide consistency among Clementine node icons.

Figure 19-6
*Background layer for all node icons*

You can create custom icons for both nodes in the palette and generated models on the stream canvas. Both are composed of the layers described here.

Table 19-1
*Composition of node and generated model icons*

<table>
<thead>
<tr>
<th></th>
<th>Node icons</th>
<th>Generated Model icons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glyph layer</td>
<td><img src="image" alt="Glyph" /></td>
<td><img src="image" alt="Glyph" /></td>
</tr>
</tbody>
</table>
Creating a Custom Icon

The following instructions pertain to the creation of a standard-sized icon. For small icons, use sample files from CEMI/images/ that are labeled with the suffix _sm. For example, *glyph_node_sm.gif* is the glyph layer for a small node icon.

- In a graphics package that supports transparency, open the example CEMI icon called *glyph_node_lg.gif* from the *CEMI/images/* directory of your Clementine installation. This is the **glyph layer**, used for a standard-sized icon.

- Using a color depth of 16 or more colors, create an icon that conveys the domain your CEMI addresses (such as text mining, micro-array analysis, etc.).

- Check the image size. Images larger than 48 pixels by 48 pixels (or 36 by 36 for small icons) will be trimmed for display in the application.

- Set the image background to transparent. In most graphics packages, transparencies are achieved by nominating a transparency color and then “painting” the image background with this color.

- If your graphics package supports layers, open a background layer from the sample directory, such as *bg_process_lg.gif*. Bring the layer into the current image.

- Check whether your custom image obscures the node background or border.

- If the layers overlap correctly, close only the background layer without saving it.

- Save the transparent glyph layer as a new *.gif* file. Note that the original example icon is read-only.
Open the CEMI specification file, and include parameters for the custom node. You should include an ICON subsection in the NODE specification (and MODEL specification if the custom icon represents a generated model node). For more information, see “ICON Subsection” in Appendix E on page 803.

Note: If your graphics package does not support layered images, individually open the glyph and background layers anyway to help visualize the correct size and placement of your custom icon.

**Generated Model Icons**

To create generated model icons for your CEMI application, repeat the steps above using the following glyph and background layers:

- For the glyph layer, use `glyph_genmod_lg.gif` for standard-sized icons.
- For testing with the background layer, use `bg_refmod_lg.gif` for refined models and `bg_unrefmod_lg.gif` for unrefined models.

**Tips for Writing External Programs**

Whether writing new programs or adapting existing programs to be called from CEMI nodes, seamless integration and ease of use should be your two main goals. With this in mind, consider the following points as you undertake the programming task:

- Programs should be able to read both symbolic and numeric data, even if the algorithms themselves can process only one type of data or more than one type of data. If this is not the case, the user of the external program will need to ensure that all field types are fully instantiated before running the program or otherwise risk invalid data getting passed to the program. The CEMI can auto-instantiate field types as long as there is a Type node upstream from the CEMI node.

- The limit on memory usage that can be set for Clementine does not apply to external programs. When writing programs, consider both the memory and speed implications of any algorithm used.
Application Examples

Overview

The data mining tools in Clementine can help solve a wide variety of business and organizational problems. The following examples are a small subset of the issues for which Clementine can provide insight.

You can use each example as a road map for the types of operations typically performed by data miners. To begin, you should load the data file(s) referenced for each application and follow the steps, learning both Clementine's visual programming interface as well as data mining methods. The data files are available from the demos directory of your installation of Clementine.

The data sets used here are much smaller than the enormous data stores managed by some data miners, but this will enable you to focus on data mining operations rather than problems with the data itself. Handling the complexities of enormous data sets and data of poor quality are discussed elsewhere in this guide. Consulting the Clementine Application Templates (CATs), available on a separate CD from your SPSS representative, will also provide a step-by-step guide to complex data mining applications.

Condition Monitoring Example

This example concerns monitoring status information from a machine and the problem of recognizing and predicting fault states. The data consist of a number of concatenated time series. Each record is a “snapshot” report on the machine in terms of the following:

- *Power*. An integer.
Pressure. 0 if normal, 1 for a momentary pressure warning.

Uptime. Time since last serviced.

Status. Normally 0, changes to error code on error (101, 202, or 303).

Outcome. The error code that appears in this time series, or 0 if no error occurs.
(These codes are available only with the benefit of hindsight.)

For each time series, there is a series of records from a period of normal operation, followed by a period leading to the fault, as shown in the following table:

<table>
<thead>
<tr>
<th>Time</th>
<th>Power</th>
<th>Temperature</th>
<th>Pressure</th>
<th>Uptime</th>
<th>Status</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1059</td>
<td>259</td>
<td>0</td>
<td>404</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1059</td>
<td>259</td>
<td>0</td>
<td>404</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>1059</td>
<td>259</td>
<td>0</td>
<td>404</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td>1059</td>
<td>259</td>
<td>0</td>
<td>404</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>53</td>
<td>1007</td>
<td>259</td>
<td>0</td>
<td>404</td>
<td>0</td>
<td>303</td>
</tr>
<tr>
<td>54</td>
<td>998</td>
<td>259</td>
<td>0</td>
<td>404</td>
<td>0</td>
<td>303</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>839</td>
<td>259</td>
<td>0</td>
<td>404</td>
<td>0</td>
<td>303</td>
</tr>
<tr>
<td>90</td>
<td>834</td>
<td>259</td>
<td>0</td>
<td>404</td>
<td>303</td>
<td>303</td>
</tr>
<tr>
<td>0</td>
<td>965</td>
<td>251</td>
<td>0</td>
<td>209</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>965</td>
<td>251</td>
<td>0</td>
<td>209</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>965</td>
<td>251</td>
<td>0</td>
<td>209</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td>965</td>
<td>251</td>
<td>0</td>
<td>209</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>53</td>
<td>938</td>
<td>251</td>
<td>0</td>
<td>209</td>
<td>0</td>
<td>101</td>
</tr>
<tr>
<td>54</td>
<td>936</td>
<td>251</td>
<td>0</td>
<td>209</td>
<td>0</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>208</td>
<td>644</td>
<td>251</td>
<td>0</td>
<td>209</td>
<td>0</td>
<td>101</td>
</tr>
<tr>
<td>209</td>
<td>640</td>
<td>251</td>
<td>0</td>
<td>209</td>
<td>101</td>
<td>101</td>
</tr>
</tbody>
</table>

This data, created using a simulation, is purely fictitious.

The following process is common to most data mining projects:
- Examine the data to determine which attributes may be relevant to the prediction or recognition of the states of interest.
- Retain those attributes (if already present), or derive and add them to the data, if necessary.
- Use the resultant data to train rules and neural nets.
- Test the trained systems using independent test data.

**Examining the Data**

The file `condplot.str` illustrates the first part of the process. It contains the stream shown below, which plots a number of graphs. If the time series of temperature or power contains patterns that are visible to the eye, you could differentiate between impending error conditions or possibly predict their occurrence. For both temperature and power, the stream below plots the time series associated with the three different error codes on separate graphs, yielding six graphs. Select nodes separate the data associated with the different error codes.

Figure 20-1
*condplot stream*
The results of this stream are shown in the following figure.

**Figure 20-2**

*Temperature and power over time*

The graphs clearly display patterns distinguishing 202 errors from 101 and 303 errors. The 202 errors show rising temperature and fluctuating power over time; the other errors do not. However, patterns distinguishing 101 from 303 errors are less clear. Both errors show even temperature and a drop in power, but the drop in power seems steeper for 303 errors.
Based on these graphs, it appears that the presence and rate of change for both temperature and power, as well as the presence and degree of fluctuation, are relevant to predicting and distinguishing faults. These attributes should therefore be added to the data before applying the learning systems.

**Data Preparation**

Based on the results of exploring the data, the stream `condlearn.str` derives the relevant data and learns to predict faults.

**Figure 20-3**

```
condlearn stream
```

The sequence of nodes is as follows:

- **Variable File node.** Reads data file `COND1n`.
- **Derive Pressure Warnings.** Counts the number of momentary pressure warnings. Reset when time returns to 0.
- **Derive TempInc.** Calculates momentary rate of temperature change using `@DIFF1`.
- **Derive PowerInc.** Calculates momentary rate of power change using `@DIFF1`.
Derive PowerFlux. A flag, true if power varied in opposite directions in the last record and this one; that is, for a power peak or trough.

Derive PowerState. A state that starts as Stable and switches to Fluctuating when two successive power fluxes are detected. Switches back to stable only when there hasn't been a power flux for five time intervals or when Time is reset.

PowerChange. Average of PowerInc over the last five time intervals.

TempChange. Average of TempInc over the last five time intervals.

Discard Initial (select). Discards the first record of each time series to avoid large (incorrect) jumps in Power and Temperature at boundaries.

Discard fields. Cuts records down to Uptime, Status, Outcome, Pressure Warnings, PowerState, PowerChange, and TempChange.

Type. Defines the direction of Outcome as Out (the field to predict). In addition, defines the type of Outcome as Auto Symbol, Pressure Warnings as Auto Number, and PowerState as Flag.

Learning

Executing the stream in condlearn.str trains the C5.0 rule and neural network (net). The network may take some time to train, but training can be interrupted early to save a net that produces reasonable results. Once the learning is complete, the Models tab at the upper right of the managers window flashes to alert you that two new nodes were created: one represents the neural net and one represents the rule.

Figure 20-4
Models manager with generated nodes
You can add generated model nodes to the existing stream to test the system or export the results of the model. In this example, we will test the results of the model.

**Testing**

Once the generated model nodes are added to the stream, a Type node is inserted and connects to the generated neural net; the net connects to the generated rule; and the rule connects to a new Analysis node. The original source node is then edited to read the file $COND2n$ (instead of $COND1n$), which contains unseen test data.

*Figure 20-5 Testing the trained network*

Executing the Analysis node yields figures reflecting the accuracy of the trained network and rule.
Fraud Detection Example

This example shows the use of Clementine in detecting behavior that might indicate fraud. The domain concerns applications for agricultural development grants, in which a data record describes a single farm's application for a particular type of grant. Two grant types are considered: arable development and decommissioning of land.

In particular, the example uses fictitious data to demonstrate the use of neural networks to detect deviations from the norm, highlighting those records that are abnormal and worthy of further investigation. You are primarily interested in grant applications that appear to claim too much money for the type and size of farm.

An overview of the operations conducted in this stream follows.

Figure 20-6
Stream diagram illustrating the operations of fraud.str stream

For this example, we'll work step-by-step, from accessing data through training a neural net.
**Accessing the Data**

The first step is to connect to the data set `grantfraudN.db` using a Variable File node. Since the data set contains field names, we can add a Table node to the stream and execute it in order to inspect its form. Alternatively, you can also gain some initial insight into the data by clicking the Types tab of the Source node and reading in the values.

The data contain nine fields:

- **id.** A unique identification number.
- **name.** Name of the claimant.
- **region.** Geographic location (*midlands/north/southwest/southeast*).
- **landquality.** Integer scale—farmer's declaration of land quality.
- **rainfall.** Integer scale—annual rainfall over farm.
- **farmincome.** Real range—declared annual income of farm.
- **maincrop.** Primary crop (*maize/wheat/potatoes/rapeseed*).
- **claimtype.** Type of grant applied for (*decommission_land/arable_dev*).
- **claimvalue.** Real range—the value of the grant applied for.

**Data Investigation**

At this point, it's a good idea to investigate the data using exploratory graphics. This helps you to form hypotheses that can be useful in modeling. To begin with, consider the possible types of fraud in the data. One such possibility is multiple grant aid applications from a single farm. Assuming that the data set contains one unique identification number per farm, it is a simple matter to show the number of occurrences of this supposedly unique number.

Connect a Distribution node to the data set and select the **name** field. The table below shows that a few farms have made multiple claims.
To explore other fraud possibilities, you can discard multiple records and focus on the characteristics of a single farm applying for aid. Records are discarded in Clementine using the Select node.
Using Clementine, you can build a model for estimating what you would expect a farm's income to be, based on its size, main crop type, soil type, and so on. To prepare for modeling, you need to derive new fields using the CLEM language in a Derive node. Use the following as your estimation of farm income:

**Figure 20-9**
*Estimating farm income*

To investigate those farmers who deviate from the estimate, you need to derive another field that compares the two values and returns a percentage difference; this field will be called `diff`.
To explore the deviations, it is helpful to plot a histogram of $diff$. It is interesting to overlay $claimtype$ to see if this has any influence on distance from the estimated income.
All of the large deviations seem to occur for \textit{arable\_dev} grants. On the basis of this, it is worth selecting only \textit{arable\_dev} grant applications. To do so, attach a Select node to the Derive node called \textit{diff} and select records using the CLEM expression \\textit{claimtype} == 'arable\_dev'.

\textbf{Training a Neural Network}

Given the initial data exploration, it seems useful to compare the actual value of claims with the value one might expect given a variety of factors. This is where a neural network can help. Using the variables in your data, the neural net can make a prediction based on the target, or dependent, variable. Using these predictions you can explore records or groups of records that deviate.

In preparation for modeling, you should first attach a Type node to the current stream. Since you want to predict the claim value using other variables in the data, the Type node can be used to set the direction of \textit{claimvalue} to \textit{Out}. 
Attach a Neural Net node and execute. Once the net has been trained, add the generated model to the stream and plot a graph of predicted claim value against actual claim value.
The fit appears to be good for the majority of cases. Derive another `claimdiff` field, similar to the “income differences” field derived earlier. This Derive node uses the CLEM expression

\[
\frac{\text{abs}(\text{claimvalue} - \$N-\text{claimvalue})}{\text{claimvalue}} \times 100
\]

In order to interpret the difference between actual and estimated claim values, use a histogram of `claimdiff`. You are primarily interested in those who appear to be claiming more than you would expect (as judged by the neural net).
By adding a band to the histogram, you can right-click in the banded area and generate a Select node to further investigate those with a relatively large `claimdiff`, such as greater than 50%. These claims warrant further investigation.

**Summary**

This example created a model and compared the model predictions to values existing in the data set (for farm incomes). From this, you found deviations mainly in one type of grant application (arable development) and selected these for further investigation. You trained a neural network model to generalize the relations between claim value and farm size, estimated income, main crop, etc. The claims that differed by a large amount from the network model (more than 50%) were identified as worth further investigation. Of course, it may turn out that all of these claims are valid, but the fact that they are different from the norm is of interest.
Retail Example

This example deals with data that describe retail product lines and the effects of promotion on sales. (This data is purely fictitious.) Your goal in this example is to predict the effects of future sales promotions. Similar to the condition monitoring example, the data mining process consists of exploration, data preparation, training, and test phases.

Examining the Data

Each record contains:

- **Class.** Product type.
- **Price.** Unit price.
- **Promotion.** Index of amount spent on a particular promotion.
- **Before.** Revenue before promotion.
- **After.** Revenue after promotion.
The stream `goods.str` contains a simple stream to display the data, producing the table in the following figure:

**Figure 20-15**

*Effects of promotion on product sales*

<table>
<thead>
<tr>
<th>Class</th>
<th>Cost</th>
<th>Promotion</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confection</td>
<td>50.590</td>
<td>1753</td>
<td>251267</td>
<td>272476</td>
</tr>
<tr>
<td>Drink</td>
<td>70.980</td>
<td>1483</td>
<td>182014</td>
<td>198040</td>
</tr>
<tr>
<td>Meat</td>
<td>36.360</td>
<td>1016</td>
<td>238983</td>
<td>248089</td>
</tr>
<tr>
<td>Luxury</td>
<td>7.340</td>
<td>1869</td>
<td>274242</td>
<td>298699</td>
</tr>
<tr>
<td>Confection</td>
<td>5.080</td>
<td>1004</td>
<td>214548</td>
<td>227949</td>
</tr>
<tr>
<td>Drink</td>
<td>62.700</td>
<td>1872</td>
<td>126390</td>
<td>144536</td>
</tr>
<tr>
<td>Meat</td>
<td>74.840</td>
<td>1914</td>
<td>130946</td>
<td>133915</td>
</tr>
<tr>
<td>Confection</td>
<td>30.630</td>
<td>1368</td>
<td>126319</td>
<td>138184</td>
</tr>
<tr>
<td>Meal</td>
<td>65.310</td>
<td>1860</td>
<td>197340</td>
<td>206279</td>
</tr>
<tr>
<td>Luxury</td>
<td>63.900</td>
<td>1110</td>
<td>165443</td>
<td>190111</td>
</tr>
<tr>
<td>Confection</td>
<td>87.760</td>
<td>1251</td>
<td>126310</td>
<td>132835</td>
</tr>
<tr>
<td>Drink</td>
<td>49.540</td>
<td>1766</td>
<td>259893</td>
<td>294010</td>
</tr>
<tr>
<td>Meal</td>
<td>65.850</td>
<td>1770</td>
<td>259752</td>
<td>267969</td>
</tr>
<tr>
<td>Luxury</td>
<td>73.330</td>
<td>1254</td>
<td>172223</td>
<td>180541</td>
</tr>
<tr>
<td>Confection</td>
<td>68.580</td>
<td>1466</td>
<td>163430</td>
<td>175810</td>
</tr>
<tr>
<td>Meal</td>
<td>81.200</td>
<td>1038</td>
<td>250355</td>
<td>256520</td>
</tr>
<tr>
<td>Meal</td>
<td>36.620</td>
<td>1077</td>
<td>134138</td>
<td>136527</td>
</tr>
<tr>
<td>Drink</td>
<td>53.590</td>
<td>1251</td>
<td>293472</td>
<td>318093</td>
</tr>
<tr>
<td>Drink</td>
<td>103.140</td>
<td>1410</td>
<td>204856</td>
<td>226596</td>
</tr>
<tr>
<td>Confection</td>
<td>104.980</td>
<td>1427</td>
<td>171759</td>
<td>182724</td>
</tr>
<tr>
<td>Luxury</td>
<td>50.640</td>
<td>1255</td>
<td>273083</td>
<td>288435</td>
</tr>
</tbody>
</table>

The two revenue fields (*Before* and *After*) are expressed in absolute terms; however, it seems likely that the increase in revenue after the promotion (and presumably as a result of it) would be a more useful figure.

The stream `goodsplot.str` derives this value, expressed as a percentage of the revenue before the promotion, in a field called *Increase* and displays a table showing this field.
In addition, the stream displays a histogram of the increase and a scatterplot of the increase against the promotion costs expended, overlaid with the category of product involved.
The scatterplot shows that for each class of product, an almost linear relationship exists between the increase in revenue and the cost of promotion. Therefore, it seems likely that a decision tree or neural network could predict, with reasonable accuracy, the increase in revenue from the other fields available.
Learning and Testing

The stream goodslearn.str trains a neural network and a decision tree to make this prediction of revenue increase.
Once you have executed the model nodes and generated the actual models, you can test the results of the learning process. You do this by connecting the decision tree and network in series between the Type node and a new Analysis node, changing the input (data) file to GOODS2n, and executing the Analysis node. From the output of this node, in particular from the linear correlation between the predicted increase and the correct answer, you will find that the trained systems predict the increase in revenue with a high degree of success.

Further exploration could focus on the cases where the trained systems make relatively large errors; these could be identified by plotting the predicted increase in revenue against the actual increase. Outliers on this graph could be selected using Clementine's interactive graphics, and from their properties, it might be possible to tune the data description or learning process to improve accuracy.

**Market Basket Analysis Example**

This example deals with fictitious data describing the contents of supermarket “baskets” (that is, collections of items bought together), plus the associated personal data of the purchaser, which might be acquired through a “loyalty card” scheme.
The goal is to discover groups of customers who buy similar products and can be characterized demographically, such as by age, income, and so on.

This example illustrates two phases of data mining:

- Association rule modeling and a web display revealing links between items purchased.
- C5.0 rule induction profiling the purchasers of identified product groups.

Unlike the other examples in this guide, this application does not make direct use of predictive modeling, so there is no accuracy measurement for the resulting models and no associated training/test distinction in the data mining process. This stream also assumes that you are more familiar with the stream-building process at this point and does not immediately provide the name of the data stream used. You should follow the steps to create your own stream and check it with the demo streams referenced periodically in the example.

### Accessing the Data

Using a Variable File node, connect to the data set `BASKETS1n`, selecting to read field names from the file. Connect a Type node to the data source, and then connect the node to a Table node. Set the type of the field `cardid` to `Typeless` (because each loyalty card ID occurs only once in the data set and can therefore be of no use in modeling). Select `Set` as the type for the field `sex` (this is to ensure that the GRI modeling algorithm will not treat `sex` as a flag). The file `bask.str` contains the stream constructed so far.

Figure 20-20

Now execute the stream to instantiate the Type node and display the table. The data set contains 18 fields, with each record representing a “basket.” The 18 fields are presented in the following headings.
Basket summary:

- *cardid*. Loyalty card identifier for customer purchasing this basket.
- *value*. Total purchase price of basket.
- *pmethod*. Method of payment for basket.

Personal details of cardholder:

- *sex*
- *homeown*. Whether or not cardholder is a homeowner.
- *income*
- *age*

Basket contents—flags for presence of product categories:

- *fruitveg*
- *freshmeat*
- *dairy*
- *cannedveg*
- *cannedmeat*
- *frozenmeal*
- *beer*
- *wine*
- *softdrink*
- *fish*
- *confectionery*

Discovering Affinities in Basket Contents

First, you need to acquire an overall picture of affinities (associations) in the basket contents using Generalized Rule Induction (GRI) to produce association rules. Select the fields to be used in this modeling process by editing the Type node and setting the directions of all of the product categories to *Both* and setting all other directions to *None*. (*Both* means that the field can be either an input or an output of the resultant model.)
Note: You can set options for multiple fields using Shift-click to select the fields before specifying an option from the columns.

Figure 20-21
Selecting fields for modeling

Once you have specified fields for modeling, attach a GRI node to the Type node, edit it, select the option Only true values for flags, and execute the GRI node. The result, an unrefined model on the Models tab at the upper right of the managers window, contains association rules that you can view using the context menu and selecting Browse.
These rules show a variety of associations between frozen meals, canned vegetables, and beer; wine and confectionery are also associated. The presence of two-way association rules, such as:

- frozenmeal $\subseteq$ beer
- beer $\subseteq$ frozenmeal
suggests that a web display (which shows only two-way associations) might highlight
some of the patterns in this data. Attach a Web node to the Type node, edit the Web
node, select all of the basket contents fields, select Show true flags only, and execute
the Web node. The following web display appears:

Figure 20-23
Web display of product associations

Because most combinations of product categories occur in several baskets, the strong
links on this web are too numerous to show the groups of customers suggested by
the GRI model. You need to raise the thresholds used by the web to show only the
strongest links. To select these options, use the following steps:

► Use the slider on the toolbar to show only connections of up to 50. The tooltip on
the slider gives feedback on the exact number selected.

► Then, to specify weak and strong connections, click the blue, arrow button on the
toolbar. This expands the dialog box showing the web output summary and controls.

► Select Size shows strong/normal/weak. This activates the slider controls below.

► Use the slider or specify a number in the text box to set weak links below 90.

► Use the slider or specify a number in the text box to set strong links above 100.
Applying these changes results in the following web display:

Figure 20-24
Restricted web display

In the display, three groups of customers stand out:

- Those who buy fish and fruits and vegetables, who might be called “healthy eaters”
- Those who buy wine and confectionery
- Those who buy beer, frozen meals, and canned vegetables (“beer, beans, and pizza”)

Note that GRI identified only the last two of these groups; the healthy eaters did not form a strong enough pattern for GRI to find it.

The file basklinks.str contains the stream constructed so far.

**Profiling the Customer Groups**

You have now identified three groups of customers based on the types of products they buy, but you would also like to know who these customers are—that is, their demographic profile. This can be achieved by “tagging” each customer with a flag for
each of these groups and using rule induction (C5.0) to build rule-based profiles of these flags.

First, you must derive a flag for each group. This can be autogenerated using the web display that you just created. Using the middle mouse button, select the link between fruitveg and fish; when selected, the link turns red. Right-click on the link and select Generate Derive Node For Link.

Figure 20-25
Deriving a flag for each customer group

Edit the resulting Derive node to change the field name to healthy. Repeat the exercise with the link from wine to confectionery, naming the resultant flag wine_chocs. For the third group (involving three links), first make sure that no links are selected; link selection can be toggled with the middle mouse button. Then select all three links in the cannedveg, beer, and frozenmeal triangle, and from the web display menus choose:
Generate
Derive Node ("And")

Change the name of the resultant flag to beer_beans_pizza.
To profile these customer groups, connect the existing Type node to these three Derive nodes, and then attach another Type node. In the new Type node, set all fields to direction *None*, except for *value, pmethod, sex, homeown, income, and age*, which should be set to *In*, and the relevant customer group (for example, *beer_beans_pizza*), which should be set to *Out*. Attach a C5.0 node, set the Output type to *Rule set*, and execute it. The resultant model (for *beer_beans_pizza*) contains a clear demographic profile for this customer group:

Rule 1 for T:
if income <= 16900
and sex == M
then -> T

The file *baskrule.str* contains the stream constructed so far, which looks like this:

**Figure 20-26**
*baskrule stream*

The same method can be applied to the other customer group flags by selecting them as the output in the second Type node. A wider range of alternative profiles can be generated by using GRI instead of C5.0 in this context; GRI can also be used to profile all of the customer group flags simultaneously because it is not restricted to a single output field.
Summary

This example reveals how Clementine can be used to discover affinities, or links, in a database, both by modeling (using GRI) and by visualization (using a web display). These links correspond to groupings of cases in the data, and these groups can be investigated in detail and profiled by modeling (using C5.0 rulesets).

In the retail domain, such customer groupings might, for example, be used to target special offers to improve the response rates to direct mailings or to customize the range of products stocked by a branch to match the demands of its demographic base.
CLEM Language Reference

CLEM Reference Overview

This section describes the Clementine Language for Expression Manipulation (CLEM), which is a powerful tool used to analyze and manipulate the data used in Clementine streams. You can use CLEM within nodes to perform the following tasks:

- Compare and evaluate conditions on record fields.
- Derive values for new fields.
- Derive new values for existing fields.
- Reason about the sequence of records.
- Insert data from records into reports.

CLEM expressions consist of values, field names, operators, and functions. Using correct syntax, you can create a wide variety of powerful data operations. For more information, see “CLEM Examples” in Chapter 8 on page 208.

CLEM Datatypes

CLEM datatypes may be made up of any of the following:

- Integers
- Reals
- Characters
- Strings
- Lists
- Fields
- Date/Time
**Rules for Quoting**

Although Clementine is flexible when determining the fields, values, parameters, and strings used in a CLEM expression, the following general rules provide a list of “good practices” to use when creating expressions.

- **Strings**—Always use double quotes when writing strings, such as "Type 2". Single quotes may be used instead but at the risk of confusion with quoted fields.
- **Fields**—Use single quotes only where necessary to enclose spaces or other special characters, such as 'Order Number'. Fields that are quoted but undefined in the data set will be misread as strings.
- **Parameters**—Always use single quotes when using parameters, such as '$P-threshold'.
- **Characters** must use single backquotes (‘), such as stripchar('d', 'drugA').

For more information, see “Values and Data Types” in Chapter 8 on page 206. Additionally, these rules are covered in more detail in the following topics.

**Integers**

Integers are represented as a sequence of decimal digits. Optionally, you can place a minus sign (−) before the integer to denote a negative number—for example, 1234, 999, −77.

The CLEM language handles integers of arbitrary precision. The maximum integer size depends on your platform. If the values are too large to be displayed in an integer field, changing the field type to Real usually restores the value.

**Reals**

*Real* refers to a floating-point number. Reals are represented by one or more digits, followed by a decimal point, followed by one or more digits. CLEM reals are held in double precision.

Optionally, you can place a minus sign (−) before the real to denote a negative number—for example, 1.234, 0.999, −77.001. Use the form <number> e <exponent> to express a real number in exponential notation—for example, 1234.0e5, 1.7e−2.

When the Clementine application reads number strings from files and converts them
automatically to numbers, numbers with no leading digit before the decimal point or with no digit after the point are accepted—for example, 999. or .11. However, these forms are illegal in CLEM expressions.

**Characters**

Characters (usually shown as CHAR) are typically used within a CLEM expression to perform tests on strings. For example, you can use the function `isuppercode` to determine whether the first character of a string is uppercase. The following CLEM expression uses a character to indicate that the test should be performed on the first character of the string:

```
isuppercode(subscrs(1, "MyString"))
```

To express the code (in contrast to the location) of a particular character in a CLEM expression, use single backquotes of the form `\`<character>`\`—for example, `\`A\`, `\`Z\`.

*Note*: There is no CHAR storage type for a field, so if a field is derived or filled with an expression that results in a CHAR, then that result will be converted to a string.

**Strings**

Generally, you should enclose strings in double quotation marks. Examples of strings are "c35product2", "referrerID". To indicate special characters in a string, use a backslash—for example, "\$6543". You can use single quotes around a string, but the result is indistinguishable from a quoted field ("referrerID").

**Lists**

A list is an ordered sequence of elements, which may be of mixed type. Lists are enclosed in square brackets ([ ]). Examples of lists are [1 2 4 16], ["abc" "def"]. Lists are not used as the value of Clementine fields. They are used to provide arguments to functions, such as `member` and `oneof`. 
Fields

Names in CLEM expressions that are not names of functions are assumed to be field names. You can write these simply as Power, val27, state_flag, etc., but if the name begins with a digit or includes non-alphabetic characters, such as spaces (with the exception of the underscore '_'), place the name within single quotation marks—for example, 'Power Increase', '2nd answer', '#101', '$P\text{-NextField}$'.

*Note:* Fields that are quoted but undefined in the data set will be misread as strings.

Dates

The CLEM language supports the following date formats:

<table>
<thead>
<tr>
<th>Format</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDMMYY</td>
<td>150163</td>
</tr>
<tr>
<td>MMDDYY</td>
<td>011563</td>
</tr>
<tr>
<td>YYMMDD</td>
<td>630115</td>
</tr>
<tr>
<td>YYYYMMDD</td>
<td>19630115</td>
</tr>
<tr>
<td>DD/MM/YY</td>
<td>15/01/63</td>
</tr>
<tr>
<td>DD/MM/YYYY</td>
<td>15/01/1963</td>
</tr>
<tr>
<td>MM/DD/YY</td>
<td>01/15/63</td>
</tr>
<tr>
<td>MM/DD/YYYY</td>
<td>01/15/1963</td>
</tr>
<tr>
<td>DD-MM-YY</td>
<td>15-01-63</td>
</tr>
<tr>
<td>DD-MM-YYYY</td>
<td>15-01-1963</td>
</tr>
<tr>
<td>MM-DD-YY</td>
<td>01-15-63</td>
</tr>
<tr>
<td>MM-DD-YYYY</td>
<td>01-15-1963</td>
</tr>
<tr>
<td>DD.MM.YY</td>
<td>15.01.63</td>
</tr>
<tr>
<td>DD.MM.YYYY</td>
<td>15.01.1963</td>
</tr>
<tr>
<td>MM.DD.YY</td>
<td>01.15.63</td>
</tr>
<tr>
<td>MM.DD.YYYY</td>
<td>01.15.1963</td>
</tr>
<tr>
<td>DD.MM.YY</td>
<td>15.01.63</td>
</tr>
<tr>
<td>DD.MM.YYYY</td>
<td>15.01.1963</td>
</tr>
<tr>
<td>MM.DD.YY</td>
<td>01.15.63</td>
</tr>
<tr>
<td>MM.DD.YYYY</td>
<td>01.15.1963</td>
</tr>
<tr>
<td>DD-MON-YY</td>
<td>15-JAN-63, 15-jan-63, 01-Jan-63</td>
</tr>
<tr>
<td>DD/MON/YY</td>
<td>15/JAN/63, 15/jan/63, 01/Jan/63</td>
</tr>
<tr>
<td>DD.MON.YY</td>
<td>15.JAN.63, 15.jan.63, 01.Jan.63</td>
</tr>
</tbody>
</table>
Date calculations are based on a “baseline” date, which is specified in the stream properties dialog box. The default baseline date is January 1, 1900.

**Time**

The CLEM language supports the following time formats:

<table>
<thead>
<tr>
<th>Format</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHMMSS</td>
<td>120112, 010101, 221212</td>
</tr>
<tr>
<td>HHMM</td>
<td>1223, 0745, 2207</td>
</tr>
<tr>
<td>MMSS</td>
<td>5558, 0100</td>
</tr>
<tr>
<td>HH:MM:SS</td>
<td>12:01:12, 01:01:01, 22:12:12</td>
</tr>
<tr>
<td>HH:MM</td>
<td>12:23, 07:45, 22:07</td>
</tr>
<tr>
<td>MM:SS</td>
<td>55:58, 01:00</td>
</tr>
<tr>
<td>(M)M:(S)S</td>
<td>55:58, 1:0</td>
</tr>
<tr>
<td>HH.MM.SS</td>
<td>12.01.12, 01.01.01, 22.12.12</td>
</tr>
<tr>
<td>HH.MM</td>
<td>12.23, 07.45, 22.07</td>
</tr>
<tr>
<td>MM.SS</td>
<td>55.58, 01.00</td>
</tr>
<tr>
<td>(H)H.(M)M.(S)S</td>
<td>12.1.12, 1.1.1, 22.12.12</td>
</tr>
<tr>
<td>(H)H.(M)M</td>
<td>12.23, 7.45, 22.7</td>
</tr>
<tr>
<td>(M)M.(S)S</td>
<td>55.58, 1.0</td>
</tr>
</tbody>
</table>
Operator Precedence

Precedences determine the parsing of complex expressions, especially unbracketed expressions with more than one infix operator. For example,

$3 + 4 * 5$

parses as $3 + (4 * 5)$ rather than $(3 + 4) * 5$ because the relative precedences dictate that * is to be parsed before +. Every operator in the CLEM language has a precedence value associated with it; the lower this value, the more important it is on the parsing list, meaning that it will be processed sooner than other operators with lower precedence values.

Precedence values are as follows:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>10</td>
</tr>
<tr>
<td>and</td>
<td>9</td>
</tr>
<tr>
<td>=</td>
<td>7</td>
</tr>
<tr>
<td>==</td>
<td>7</td>
</tr>
<tr>
<td>/=</td>
<td>7</td>
</tr>
<tr>
<td>/=</td>
<td>7</td>
</tr>
<tr>
<td>&gt;</td>
<td>6</td>
</tr>
<tr>
<td>&gt;=</td>
<td>6</td>
</tr>
<tr>
<td>&lt;</td>
<td>6</td>
</tr>
<tr>
<td>&lt;=</td>
<td>6</td>
</tr>
<tr>
<td>&amp;&amp;=.0</td>
<td>6</td>
</tr>
<tr>
<td>&amp;&amp;/=._0</td>
<td>6</td>
</tr>
<tr>
<td>+</td>
<td>5</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>5</td>
</tr>
<tr>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td>*</td>
<td>4</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>4</td>
</tr>
<tr>
<td>&amp;&amp;~</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parameters

Parameters are effectively “variables.” They allow you to change values referred to in CLEM expressions without having to edit the expressions manually. There are three types of parameters, although they all look identical in CLEM expressions:

- **SuperNode parameters**—You can define SuperNode parameters, with default values, for any SuperNode. They are visible only to nodes encapsulated within that SuperNode.
- **Stream parameters**—These parameters are defined interactively using the Parameters tab in the stream properties dialog box. They are saved and loaded along with stream diagrams. They are cleared by clearing the stream diagram or by deleting them using the Parameters tab.
- **Session parameters**—These parameters are defined on the command line used to invoke Clementine, using arguments of the form `-P<name>=<value>`. For example, `-Pthreshold=100` defines a session parameter called `threshold` with a value of 100. In a CLEM expression, parameters are written as `$P<name>$` and must be placed within quotation marks—for example, `Price > '$P\text{-threshold}'`. 

<table>
<thead>
<tr>
<th>Operation</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>~~</code></td>
<td>4</td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>4</td>
</tr>
<tr>
<td><code>&gt;&gt;</code></td>
<td>4</td>
</tr>
<tr>
<td><code>/</code></td>
<td>4</td>
</tr>
<tr>
<td><code>**</code></td>
<td>3</td>
</tr>
<tr>
<td><code>rem</code></td>
<td>2</td>
</tr>
<tr>
<td><code>mod</code></td>
<td>2</td>
</tr>
<tr>
<td><code>div</code></td>
<td>2</td>
</tr>
</tbody>
</table>
Using Parameters in CLEM Expressions

Parameters set for SuperNodes, streams, and sessions can be accessed in CLEM. Parameters are represented in expressions by \( \$P\text{-pname} \), where \( \text{pname} \) is the name of the parameter. When used in CLEM expressions, parameters must be placed within single quotes—for example, \( \$P\text{-scale} \).

Available parameters are easily viewed using the Expression Builder. To view current parameters:

- In any dialog box accepting CLEM expressions, click the Expression Builder button.
- From the Fields drop-down list, select Parameters.

You can select parameters from the list for insertion into the CLEM expression.

Functions Reference

The following CLEM functions are available when working with data in Clementine. You can enter these functions as code in a variety of dialog boxes, such as Derive and Set To Flag nodes, or you can use the Expression Builder to create valid CLEM expressions without memorizing function lists or field names.

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information</td>
<td>Used to gain insight into field values. For example, the function ( \text{is} _\text{string} ) returns true for all records whose type is a string.</td>
</tr>
<tr>
<td>Conversion</td>
<td>Used to construct new fields or convert storage type. For example, the function ( \text{to} _\text{timestamp} ) converts the selected field to a timestamp.</td>
</tr>
<tr>
<td>Comparison</td>
<td>Used to compare field values to each other or to a specified string. For example, ( &lt;= ) is used to compare whether the values of two fields are lesser or equal.</td>
</tr>
<tr>
<td>Logical</td>
<td>Used to perform logical operations, such as if, then, else operations.</td>
</tr>
<tr>
<td>Numeric</td>
<td>Used to perform numeric calculations, such as the natural log of field values.</td>
</tr>
<tr>
<td>Trigonometric</td>
<td>Used to perform trigonometric calculations, such as the arccosine of a specified angle.</td>
</tr>
<tr>
<td>Bitwise</td>
<td>Used to manipulate integers as bit patterns.</td>
</tr>
<tr>
<td>Random</td>
<td>Used to randomly select items or generate numbers.</td>
</tr>
</tbody>
</table>
### Function Type

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Used to perform a wide variety of operations on strings, such as <code>stripchar</code>, which allows you to remove a specified character.</td>
</tr>
<tr>
<td>Date and time</td>
<td>Used to perform a variety of operations on datetime fields.</td>
</tr>
<tr>
<td>Sequence</td>
<td>Used to gain insight into the record sequence of a data set or perform operations based on that sequence.</td>
</tr>
<tr>
<td>Global</td>
<td>Used to access global values created by a Set Globals node. For example, <code>@MEAN</code> is used to refer to the mean average of all values for a field across the entire data set.</td>
</tr>
<tr>
<td>Blanks and null</td>
<td>Used to access, flag, and frequently to fill user-specified blanks or system-missing values. For example, <code>@BLANK(FIELD)</code> is used to raise a true flag for records where blanks are present.</td>
</tr>
<tr>
<td>Special fields</td>
<td>Used to denote the specific fields under examination. For example, <code>@FIELD</code> is used when deriving multiple fields.</td>
</tr>
</tbody>
</table>

### Conventions in Function Descriptions

Except in those cases where the arguments or results of a function are sufficiently complicated to require names that describe their function, rather than just their type, adhere to the following conventions:

<table>
<thead>
<tr>
<th>ITEM</th>
<th>Anything</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>A Boolean, or flag, such as true or false</td>
</tr>
<tr>
<td>NUM, NUM1, NUM2</td>
<td>Any number</td>
</tr>
<tr>
<td>REAL, REAL1, REAL2</td>
<td>Any real number</td>
</tr>
<tr>
<td>INT, INT1, INT2</td>
<td>Any integer</td>
</tr>
<tr>
<td>CHAR</td>
<td>A character code</td>
</tr>
<tr>
<td>STRING</td>
<td>A string</td>
</tr>
<tr>
<td>LIST</td>
<td>A list</td>
</tr>
<tr>
<td>ITEM</td>
<td>A field</td>
</tr>
<tr>
<td>DATE</td>
<td>A date field</td>
</tr>
<tr>
<td>TIME</td>
<td>A time field</td>
</tr>
</tbody>
</table>

Functions are shown in the format `function(argument) -> result`, where `argument` and `result` indicate types. For example, the function `sqrt(NUM)` returns a `REAL` value.
Information Functions

Information functions are used to gain insight into the values of a particular field. They are typically used to derive flag fields. For example, you can use the @BLANK function to create a flag field indicating records whose values are blank for the selected field. Similarly, you can check the storage type for a field using any of the storage type functions, such as is_string.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@BLANK(FIELD)</td>
<td>Boolean</td>
<td>Returns true for all records whose values are blank according to the blank handling rules set in an upstream Type node or Source node (Types tab).</td>
</tr>
<tr>
<td>@NULL(ITEM)</td>
<td>Boolean</td>
<td>Returns true for all records whose values are undefined. Undefined values are system null values, displayed in Clementine as $null$.</td>
</tr>
<tr>
<td>is_date(ITEM)</td>
<td>Boolean</td>
<td>Returns true for all records whose type is a date.</td>
</tr>
<tr>
<td>is_datetime(ITEM)</td>
<td>Boolean</td>
<td>Returns true for all records whose type is datetime.</td>
</tr>
<tr>
<td>is_integer(ITEM)</td>
<td>Boolean</td>
<td>Returns true for all records whose type is an integer.</td>
</tr>
<tr>
<td>is_number(ITEM)</td>
<td>Boolean</td>
<td>Returns true for all records whose type is a number.</td>
</tr>
<tr>
<td>is_real(ITEM)</td>
<td>Boolean</td>
<td>Returns true for all records whose type is a real.</td>
</tr>
<tr>
<td>is_string(ITEM)</td>
<td>Boolean</td>
<td>Returns true for all records whose type is a string.</td>
</tr>
<tr>
<td>is_time(ITEM)</td>
<td>Boolean</td>
<td>Returns true for all records whose type is time.</td>
</tr>
<tr>
<td>is_timestamp(ITEM)</td>
<td>Boolean</td>
<td>Returns true for all records whose type is a timestamp.</td>
</tr>
</tbody>
</table>

Conversion Functions

Conversion functions allow you to construct new fields and convert the storage type of existing files. For example, you can form new strings by joining strings together or by taking strings apart. To join two strings, use the operator >>. For example, if the field Site has the value "BRAMLEY", then "xx" >> Site returns "xxBRAMLEY". The result of >> is always a string, even if the arguments are not strings. Thus, if field V1 is 3 and field V2 is 5, V1 >> V2 returns "35" (a string, not a number).
**Function** | **Result** | **Description**
--- | --- | ---
ITEM1 > ITEM2 | String | Concatenates values for two fields and returns the resulting string as ITEM1 ITEM2.
to_integer(ITEM) | Integer | Converts the storage of the specified field to an integer.
to_real(ITEM) | Real | Converts the storage of the specified field to a real.
to_string(ITEM) | String | Converts the storage of the specified field to a string.
to_time(ITEM) | Time | Converts the storage of the specified field to a time.
to_date(ITEM) | Date | Converts the storage of the specified field to a date.
to_timestamp(ITEM) | Timestamp | Converts the storage of the specified field to a timestamp.

**Comparison Functions**

Comparison functions are used to compare field values to each other or to a specified string. For example, you can check strings for equality using =. An example of string equality verification is: Class = "class 1".

For purposes of numeric comparison, greater means closer to positive infinity, and lesser means closer to negative infinity. That is, all negative numbers are less than any positive number.

\[ \text{NUM1} = \text{NUM2} \rightarrow \text{BOOL} \]

**Function** | **Result** | **Description**
--- | --- | ---
ITEM1 = ITEM2 | Boolean | Returns true for records where ITEM1 is equal to ITEM2.
ITEM1 /= ITEM2 | Boolean | Returns true if the two strings are not identical and 0 if they are identical.
ITEM1 < ITEM2 | Boolean | Returns true for records where ITEM1 is less than ITEM2.
ITEM1 <= ITEM2 | Boolean | Returns true for records where ITEM1 is less than or equal to ITEM2.
ITEM1 > ITEM2 | Boolean | Returns true for records where ITEM1 is greater than ITEM2.
ITEM1 >= ITEM2 | Boolean | Returns true for records where ITEM1 is greater than or equal to ITEM2.
alphabefore(STRING1, STRING2) | Boolean | Used to check the alphabetical ordering of strings. Returns a true value if STRING1 precedes STRING2.
date_before(DATE1, DATE2) | Boolean | Used to check the ordering of date values. Returns a true value if DATE1 is before DATE2.
### Logical Functions

CLEM expressions can be used to perform logical operations.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COND1 and COND2</td>
<td>Boolean</td>
<td>This operation is a logical conjunction and returns a true value if both COND1 and COND2 are true. If COND1 is false, then COND2 is not evaluated; this makes it possible to have conjunctions where COND1 first tests that an operation in COND2 is legal. For example, length(Label) &gt;=6 and Label(6) = 'x'.</td>
</tr>
<tr>
<td>COND1 or COND2</td>
<td>Boolean</td>
<td>This operation is a logical (inclusive) disjunction and returns a true value if either COND1 or COND2 is true or if both are true. If COND1 is true, COND2 is not evaluated.</td>
</tr>
<tr>
<td>not(COND)</td>
<td>Boolean</td>
<td>This operation is a logical negation and returns a true value if COND is false. Otherwise, this operation returns a value of 0.</td>
</tr>
<tr>
<td>if COND then EXPR1 else EXPR2 endif</td>
<td>Any</td>
<td>This operation is a conditional evaluation. If COND is true, this operation returns the result of EXPR1. Otherwise, the result of evaluating EXPR2 is returned.</td>
</tr>
<tr>
<td>if COND1 then EXPR1 elseif COND2 then EXPR2 else EXPR_N endif</td>
<td>Any</td>
<td>This operation is a multibranch conditional evaluation. If COND1 is true, this operation returns the result of EXPR1. Otherwise, if COND2 is true, this operation returns the result of evaluating EXPR2. Otherwise, the result of evaluating EXPR_N is returned.</td>
</tr>
</tbody>
</table>

### Numeric Functions

CLEM contains a number of commonly used numeric functions.
<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>- NUM</td>
<td>Number</td>
<td>Used to negate NUM. Returns the corresponding number with the opposite sign.</td>
</tr>
<tr>
<td>NUM1 + NUM2</td>
<td>Number</td>
<td>Returns the sum of NUM1 and NUM2.</td>
</tr>
<tr>
<td>\text{code} - NUM2</td>
<td>Number</td>
<td>Returns the value of NUM2 subtracted from NUM1.</td>
</tr>
<tr>
<td>NUM1 * NUM2</td>
<td>Number</td>
<td>Returns the value of NUM1 multiplied by NUM2.</td>
</tr>
<tr>
<td>NUM1 / NUM2</td>
<td>Number</td>
<td>Returns the value of NUM1 divided by NUM2.</td>
</tr>
<tr>
<td>INT1 div INT2</td>
<td>Number</td>
<td>Used to perform integer division. Returns the value of INT1 divided by INT2.</td>
</tr>
<tr>
<td>INT1 rem INT2</td>
<td>Number</td>
<td>Returns the remainder of INT1 divided by INT2. For example, INT1 – ((\text{INT1 div INT2}) \times \text{INT2}).</td>
</tr>
<tr>
<td>INT1 mod INT2</td>
<td>Number</td>
<td>Similar to INT1 rem INT2, the modulo function gives the remainder when dividing INT1 by INT2. When INT1 and INT2 have the same sign, the results are the same. When INT1 and INT2 have different signs (and the remainder is not zero), the value of \text{mod} is the value of \text{rem} plus INT2.</td>
</tr>
<tr>
<td>BASE ** POWER</td>
<td>Number</td>
<td>Returns BASE raised to the power POWER, where either may be any number (except that BASE must not be zero if \text{POWER} is zero of any type other than integer 0). If \text{POWER} is an integer, the computation is performed by successively multiplying powers of BASE. Thus, if BASE is an integer, the result will be an integer. If \text{POWER} is integer 0, the result is always a 1 of the same type as BASE. Otherwise, if \text{POWER} is not an integer, the result is computed as (\exp(\text{POWER} \times \log(\text{BASE}))).</td>
</tr>
<tr>
<td>abs(NUM)</td>
<td>Number</td>
<td>Returns the absolute value of NUM, which is always a number of the same type.</td>
</tr>
<tr>
<td>exp(NUM)</td>
<td>Real</td>
<td>Returns (e) raised to the power NUM, where (e) is the base of natural logarithms.</td>
</tr>
<tr>
<td>fracof(NUM)</td>
<td>Real</td>
<td>Returns the fractional part of NUM, defined as NUM–intof(NUM).</td>
</tr>
<tr>
<td>intof(NUM)</td>
<td>Integer</td>
<td>Truncates its argument to an integer. It returns the integer of the same sign as NUM and with the largest magnitude such that abs(INT) &lt;= abs(NUM).</td>
</tr>
<tr>
<td>log(NUM)</td>
<td>Real</td>
<td>Returns the natural (base (e)) logarithm of NUM, which must not be a zero of any kind.</td>
</tr>
<tr>
<td>log10(NUM)</td>
<td>Real</td>
<td>Returns the base 10 logarithm of NUM, which must not be a zero of any kind. This function is defined as (\log(\text{NUM}) / \log(10)).</td>
</tr>
<tr>
<td>negate(NUM)</td>
<td>Number</td>
<td>Used to negate NUM. Returns the corresponding number with the opposite sign.</td>
</tr>
</tbody>
</table>
### Appendix A

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>round(NUM)</td>
<td>Integer</td>
<td>Used to round NUM to an integer by taking intof(NUM+0.5) if NUM is positive or intof(NUM–0.5) if NUM is negative.</td>
</tr>
<tr>
<td>sign(NUM)</td>
<td>Number</td>
<td>Used to determine the sign of NUM, this operation returns −1, 0, or 1 if NUM is an integer. If NUM is a real, it returns −1.0, 0.0, or 1.0, depending on whether NUM is negative, zero, or positive.</td>
</tr>
<tr>
<td>sqrt(NUM)</td>
<td>Real</td>
<td>Returns the square root of NUM. NUM must be positive.</td>
</tr>
</tbody>
</table>

### Trigonometric Functions

All of the functions in this section either take an angle as an argument or return one as a result. In both cases, the units of the angle (radians or degrees) are controlled by the setting of the relevant stream option.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arccos(NUM)</td>
<td>Real</td>
<td>Computes the arccosine of the specified angle.</td>
</tr>
<tr>
<td>arccosh(NUM)</td>
<td>Real</td>
<td>Computes the hyperbolic arccosine of the specified angle.</td>
</tr>
<tr>
<td>arcsin(NUM)</td>
<td>Real</td>
<td>Computes the arcsine of the specified angle.</td>
</tr>
<tr>
<td>arcsinh(NUM)</td>
<td>Real</td>
<td>Computes the hyperbolic arcsine of the specified angle.</td>
</tr>
<tr>
<td>arctan(NUM)</td>
<td>Real</td>
<td>Computes the arctangent of the specified angle.</td>
</tr>
<tr>
<td>arctan2(NUM_X, NUM_Y)</td>
<td>Real</td>
<td>Computes the arctangent of NUM_Y / NUM_X and uses the signs of the two numbers to derive quadrant information. The result is a real in the range - pi &lt; ANGLE &lt;= pi (radians) – 180 &lt; ANGLE &lt;= 180 (degrees)</td>
</tr>
<tr>
<td>arctanh(NUM)</td>
<td>Real</td>
<td>Computes the hyperbolic arctangent of the specified angle.</td>
</tr>
<tr>
<td>cos(NUM)</td>
<td>Real</td>
<td>Computes the cosine of the specified angle.</td>
</tr>
<tr>
<td>cosh(NUM)</td>
<td>Real</td>
<td>Computes the hyperbolic cosine of the specified angle.</td>
</tr>
<tr>
<td>pi</td>
<td>Real</td>
<td>This constant is the best real approximation to pi.</td>
</tr>
<tr>
<td>sin(NUM)</td>
<td>Real</td>
<td>Computes the sine of the specified angle.</td>
</tr>
<tr>
<td>sinh(NUM)</td>
<td>Real</td>
<td>Computes the hyperbolic sine of the specified angle.</td>
</tr>
<tr>
<td>tan(NUM)</td>
<td>Real</td>
<td>Computes the tangent of the specified angle.</td>
</tr>
<tr>
<td>tanh(NUM)</td>
<td>Real</td>
<td>Computes the hyperbolic tangent of the specified angle.</td>
</tr>
</tbody>
</table>
Bitwise Integer Operations

These functions enable integers to be manipulated as bit patterns representing two’s-complement values, where bit position $N$ has weight $2^{*N}$. Bits are numbered from 0 upward. These operations act as though the sign bit of an integer is extended indefinitely to the left. Thus, everywhere above its most significant bit, a positive integer has 0 bits and a negative integer has 1 bit.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sim$ INT1</td>
<td>Integer</td>
<td>Produces the bitwise complement of the integer $INT1$. That is, there is a 1 in the result for each bit position for which $INT1$ has 0. It is always true that $\sim$ INT = $-(INT + 1)$.</td>
</tr>
<tr>
<td>INT1</td>
<td></td>
<td>INT2</td>
</tr>
<tr>
<td>INT1</td>
<td></td>
<td>/&amp; INT2</td>
</tr>
<tr>
<td>INT1 &amp;&amp; INT2</td>
<td>Integer</td>
<td>Produces the bitwise “and” of the integers $INT1$ and $INT2$. That is, there is a 1 in the result for each bit position for which there is a 1 in both $INT1$ and $INT2$.</td>
</tr>
<tr>
<td>INT1 &amp;&amp;~ INT2</td>
<td>Integer</td>
<td>Produces the bitwise “and” of $INT1$ and the bitwise complement of $INT2$. That is, there is a 1 in the result for each bit position for which there is a 1 in $INT1$ and a 0 in $INT2$. This is the same as $INT1$ &amp;&amp; ($\sim$INT2) and is useful for clearing bits of $INT1$ set in $INT2$.</td>
</tr>
<tr>
<td>INT &lt;&lt; N</td>
<td>Integer</td>
<td>Produces the bit pattern of $INT1$ shifted left by $N$ positions. A negative value for $N$ produces a right shift.</td>
</tr>
<tr>
<td>INT &gt;&gt; N</td>
<td>Integer</td>
<td>Produces the bit pattern of $INT1$ shifted right by $N$ positions. A negative value for $N$ produces a left shift.</td>
</tr>
<tr>
<td>INT1 &amp;&amp;=_0 INT2</td>
<td>Boolean</td>
<td>Equivalent to the Boolean expression $INT1$ &amp;&amp; $INT2$ $/=0$ but is more efficient.</td>
</tr>
<tr>
<td>INT1 &amp;&amp;/= _0 INT2</td>
<td>Boolean</td>
<td>Equivalent to the Boolean expression $INT1$ &amp;&amp; $INT2$ $==0$ but is more efficient.</td>
</tr>
</tbody>
</table>
## Appendix A

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>integer_bitcount(INT)</code></td>
<td><code>Integer</code></td>
<td>Counts the number of 1 or 0 bits in the two's-complement representation of <code>INT</code>. If <code>INT</code> is non-negative, <code>N</code> is the number of 1 bits. If <code>INT</code> is negative, it is the number of 0 bits. Owing to the sign extension, there are an infinite number of 0 bits in a non-negative integer or 1 bits in a negative integer. It is always the case that ( \text{integer_bitcount}(INT) = \text{integer_bitcount}(-(INT+1)) ).</td>
</tr>
<tr>
<td><code>integer_leastbit(INT)</code></td>
<td><code>Integer</code></td>
<td>Returns the bit position <code>N</code> of the least-significant bit set in the integer <code>INT</code>. <code>N</code> is the highest power of 2 by which <code>INT</code> divides exactly.</td>
</tr>
<tr>
<td><code>integer_length(INT)</code></td>
<td><code>Integer</code></td>
<td>Returns the length in bits of <code>INT</code> as a two's-complement integer. That is, <code>N</code> is the smallest integer such that <code>INT &lt; (1 &lt;&lt; N)</code> if <code>INT &gt;= 0</code> <code>INT &gt;= (-1 &lt;&lt; N)</code> if <code>INT &lt; 0</code>. If <code>INT</code> is non-negative, then the representation of <code>INT</code> as an unsigned integer requires a field of at least <code>N</code> bit. Alternatively, a minimum of <code>N+1</code> bits is required to represent <code>INT</code> as a signed integer, regardless of its sign.</td>
</tr>
<tr>
<td><code>testbit(INT, N)</code></td>
<td><code>Boolean</code></td>
<td>Tests the bit at position <code>N</code> in the integer <code>INT</code> and returns the state of bit <code>N</code> as a Boolean value, which is true for 1 and false for 0.</td>
</tr>
</tbody>
</table>

### Random Functions

The following functions are used to randomly select items or randomly generate numbers.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oneof(LIST)</code></td>
<td><code>Any</code></td>
<td>Returns a randomly chosen element of <code>LIST</code>. List items should be entered as <code>[ITEM1,ITEM2,...ITEM_N]</code>.</td>
</tr>
<tr>
<td><code>random(NUM)</code></td>
<td><code>Number</code></td>
<td>Returns a uniformly distributed random number of the same type (INT or REAL), starting from 1 to NUM. If you use an integer, then only integers are returned. If you use a real (decimal) number, then real numbers are returned (decimal precision determined by the stream options). The largest random number returned by the function could equal NUM.</td>
</tr>
<tr>
<td><code>random0(NUM)</code></td>
<td><code>Number</code></td>
<td>This has the same properties as <code>random(NUM)</code>, but starting from 0. The largest random number returned by the function will never equal X.</td>
</tr>
</tbody>
</table>
**String Functions**

In CLEM, you can perform the following operations with strings:

- Compare strings.
- Create strings.
- Access characters.

In a CLEM expression, a string is any sequence of characters between matching double quotation marks ("string quotes"). Characters (CHAR) can be any single alphanumeric character. They are declared in CLEM expressions using single backquotes in the form of `<character>` such as `'2`, `'A`, or `'z`. Characters that are out of bounds or negative indices to a string will results in a null value.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allbutfirst(N, STRING)</td>
<td>String</td>
<td>Returns a string, which is STRING with the first N characters removed.</td>
</tr>
<tr>
<td>allbutlast(N, STRING)</td>
<td>String</td>
<td>Returns a string, which is STRING with the last N characters removed.</td>
</tr>
<tr>
<td>alphabefore(STRING1, STRING2)</td>
<td>Boolean</td>
<td>Used to check the alphabetical ordering of strings. Returns true if STRING1 precedes STRING2.</td>
</tr>
<tr>
<td>hasendstring(STRING, SUBSTRING)</td>
<td>Integer</td>
<td>This function is the same as isendstring(SUB_STRING, STRING).</td>
</tr>
<tr>
<td>hasmidstring(STRING, SUBSTRING)</td>
<td>Integer</td>
<td>This function is the same as ismidstring(SUB_STRING, STRING) (embedded substring).</td>
</tr>
<tr>
<td>hasstartstring(STRING, SUBSTRING)</td>
<td>Integer</td>
<td>This function is the same as isstartstring(SUB_STRING, STRING).</td>
</tr>
<tr>
<td>hassubstring(STRING, N, SUBSTRING)</td>
<td>Integer</td>
<td>This function is the same as issubstring(SUB_STRING, N, STRING) where N defaults to 1.</td>
</tr>
<tr>
<td>hassubstring(STRING, SUBSTRING)</td>
<td>Integer</td>
<td>This function is the same as issubstring(SUB_STRING, 1, STRING) where N defaults to 1.</td>
</tr>
<tr>
<td>isalphacode(CHAR)</td>
<td>Boolean</td>
<td>Returns a value of true if CHAR is a character in the specified string (often a field name) whose character code is a letter. Otherwise, this function returns a value of 0. For example, isalphacode(produce_num(1)).</td>
</tr>
</tbody>
</table>
## Appendix A

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isendstring(SUBSTRING, STRING)</td>
<td>Integer</td>
<td>If the string STRING ends with the substring SUBSTRING, then this function returns the integer subscript of SUBSTRING in STRING. Otherwise, this function returns a value of 0.</td>
</tr>
<tr>
<td>islowercode(CHAR)</td>
<td>Boolean</td>
<td>Returns a value of true if CHAR is a lowercase letter character for the specified string (often a field name). Otherwise, this function returns a value of 0. For example, both islowercode(‘a’) —&gt; T and islowercode(country_name(2)) —&gt; T are valid expressions.</td>
</tr>
<tr>
<td>ismidstring(SUBSTRING, STRING)</td>
<td>Integer</td>
<td>If SUB_STRING is a substring of STRING but does not start on the first character of STRING or end on the last, then this function returns the subscript at which the substring starts. Otherwise, this function returns a value of 0.</td>
</tr>
<tr>
<td>isnumbercode(CHAR)</td>
<td>Boolean</td>
<td>Returns a value of true if CHAR for the specified string (often a field name) is a character whose character code is a digit. Otherwise, this function returns a value of 0. For example, isnumbercode(product_id(2)).</td>
</tr>
<tr>
<td>isstartstring(SUBSTRING, STRING)</td>
<td>Integer</td>
<td>If the string STRING starts with the substring SUB_STRING, then this function returns the subscript 1. Otherwise, this function returns a value of 0.</td>
</tr>
<tr>
<td>issubstring(SUBSTRING, N, STRING)</td>
<td>Integer</td>
<td>Searches the string STRING, starting from its Nth character, for a substring equal to the string SUB_STRING. If found, this function returns the integer subscript at which the matching substring begins. Otherwise, this function returns a value of 0. If N is not given, this function defaults to 1.</td>
</tr>
<tr>
<td>issubstring(SUBSTRING, STRING)</td>
<td>Integer</td>
<td>Searches the string STRING, starting from its Nth character, for a substring equal to the string SUB_STRING. If found, this function returns the integer subscript at which the matching substring begins. Otherwise, this function returns a value of 0. If N is not given, this function defaults to 1.</td>
</tr>
<tr>
<td>Function</td>
<td>Result</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------------</td>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>issubstring_lim(SUBSTRING, N, STARTLIM, ENDLIM, STRING)</td>
<td>Integer</td>
<td>This function is the same as <code>issubstring</code>, but the match is constrained to start on or before the subscript <code>STARTLIM</code> and to end on or before the subscript <code>ENDLIM</code>. The <code>STARTLIM</code> or <code>ENDLIM</code> constraints may be disabled by supplying a value of false for either argument—for example, <code>issubstring_lim(SUBSTRING, N, false, false, STRING)</code> is the same as <code>issubstring</code>.</td>
</tr>
<tr>
<td>isuppercode(CHAR)</td>
<td>Boolean</td>
<td>Returns a value of true if <code>CHAR</code> is an uppercase letter character. Otherwise, this function returns a value of 0. For example, both <code>isuppercode(\'\')</code> —&gt; T and <code>isuppercode(country_name(2))</code> —&gt; T are valid expressions.</td>
</tr>
<tr>
<td>last(CHAR)</td>
<td>String</td>
<td>Returns the last character <code>CHAR</code> of <code>STRING</code>(which must be at least one character long).</td>
</tr>
<tr>
<td>length(STRING)</td>
<td>Integer</td>
<td>Returns the length of the string <code>STRING</code>—that is, the number of characters in it.</td>
</tr>
<tr>
<td>locchar(CHAR, N, STRING)</td>
<td>Integer</td>
<td>Used to identify the location of characters in symbolic fields. The function searches the string <code>STRING</code> for the character <code>CHAR</code>, starting the search at the <code>N</code>th character of <code>STRING</code>. This function returns a value indicating the location (starting at <code>N</code>) where the character is found. If the character is not found, this function returns a value of 0. If the function has an invalid offset (<code>N</code>) (for example, an offset that is beyond the length of the string), this function returns <code>null</code>. For example, <code>locchar(\'n\', 2, web_page)</code> searches the field called <code>web_page</code> for the ‘n’ character beginning at the second character in the field value. Note: Be sure to use single backquotes to encapsulate the specified character.</td>
</tr>
<tr>
<td>Function</td>
<td>Result</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>locchar_back(CHAR, N, STRING)</td>
<td>Integer</td>
<td>Similar to locchar, except that the search is performed <strong>backward</strong>, starting from the <strong>Nth</strong> character. For example,locchar_back('n', 9, web_page) searches the field web_page starting from the ninth character and moving backwards towards the start of the string. If the function has an invalid offset (for example, an offset that is beyond the length of the string), this function returns $null$. Ideally, you should use locchar_back in conjunction with the function length(&lt;field&gt;) to dynamically use the length of the current value of the field. For example, locchar_back('n', (length(web_page)), web_page).</td>
</tr>
<tr>
<td>stripchar(CHAR,STRING)</td>
<td>String</td>
<td>Enables you to remove specified characters from a string or field. You can use this function, for example, to remove extra symbols, such as currency notations, from data to achieve a simple number or name. For example, using the syntax stripchar('$', 'Cost') returns a new field with the dollar sign removed from all values. <strong>Note:</strong> Be sure to use single backquotes to encapsulate the specified character.</td>
</tr>
<tr>
<td>skipchar(CHAR, N, STRING)</td>
<td>Integer</td>
<td>Searches the string <strong>STRING</strong> for any character other than <strong>CHAR</strong>, starting at the <strong>Nth</strong> character. This function returns an integer substring indicating the point at which one is found or 0 if every character from the <strong>Nth</strong> onward is a <strong>CHAR</strong>. If the function has an invalid offset (for example, an offset that is beyond the length of the string), this function returns $null$. locchar is often used in conjunction with the skipchar functions to determine the value of <strong>N</strong> (the point at which to start searching the string). For example, skipchar('s', (locchar('s', 1, 'MyString')), 'MyString').</td>
</tr>
<tr>
<td>skipchar_back(CHAR, N, STRING)</td>
<td>Integer</td>
<td>Similar to skipchar, except that the search is performed backward, starting from the <strong>Nth</strong> character.</td>
</tr>
<tr>
<td>Function</td>
<td>Result</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>strmember(CHAR, STRING)</td>
<td>Integer</td>
<td>Equivalent to locchar(CHAR, 1, STRING). It returns an integer substring indicating the point at which CHAR first occurs, or 0. If the function has an invalid offset (for example, an offset that is beyond the length of the string), this function returns $null$.</td>
</tr>
<tr>
<td>subscrs(N, STRING)</td>
<td>CHAR</td>
<td>Returns the Nth character CHAR of the input string STRING. This function can also be written in a shorthand form—STRING(N) -&gt; CHAR. For example, lowertoupper(&quot;name&quot;(1)) is a valid expression.</td>
</tr>
<tr>
<td>substring(N, LEN, STRING)</td>
<td>String</td>
<td>Returns a string SUB_STRING, which consists of the LEN characters of the string STRING, starting from the character at subscript N.</td>
</tr>
<tr>
<td>substring_between(N1, N2, STRING)</td>
<td>String</td>
<td>Returns the substring of STRING, which begins at subscript N1 and ends at subscript N2.</td>
</tr>
</tbody>
</table>
| uppertolower(CHAR) uppertolower (STRING) | CHAR or String | Input can be either a string or character, and is used in this function to return a new item of the same type, with any uppercase characters converted to their lowercase equivalents.  
  Note: Remember to specify strings with double quotes and characters with single backquotes. Simple field names should appear without quotes. |
| lowertoupper(CHAR) lowertoupper (STRING) | CHAR or String | Input can be either a string or character, which is used in this function to return a new item of the same type, with any lowercase characters converted to their uppercase equivalents.  
  For example, lowertoupper(\`a\`). lowertoupper("My string") and lowertoupper(field_name(2)) are all valid expressions. |
Appendix A

**Date and Time Functions**

CLEM includes a family of functions for handling fields with datetime storage of string variables representing dates and times. The formats of date and time used are specific to each stream and are specified in the stream properties dialog box. The date and time functions parse date and time strings according to the currently selected format.

When you specify a year in a date that uses only two digits (that is, the century is not specified), Clementine uses the default century that is specified in the stream properties dialog box.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@TODAY</td>
<td>String</td>
<td>If you select Rollover days/mins in the stream properties dialog box, this function returns the current date as a string in the current date format. If you use a two-digit date format and do not select Rollover days/mins, this function returns $null$ on the current server.</td>
</tr>
<tr>
<td>date_before(DATE1, DATE2)</td>
<td>Boolean</td>
<td>Returns a value of true if DATE1 represents a date before that represented by DATE2. Otherwise, this function returns a value of 0.</td>
</tr>
<tr>
<td>date_days_difference(DATE1, DATE2)</td>
<td>Integer</td>
<td>Returns the time in days from the date represented by DATE1 to the date represented by DATE2, as an integer. If DATE2 is before DATE1, this function returns a negative number.</td>
</tr>
<tr>
<td>date_in_days(DATE)</td>
<td>Integer</td>
<td>Returns the time in days from the baseline date to the date represented by DATE, as an integer. If DATE is before the baseline date, this function returns a negative number. You must include a valid date for the calculation to work appropriately. For example, you should not specify February 29, 2001, as the date. Because 2001 is a not a leap year, this date does not exist.</td>
</tr>
<tr>
<td>date_in_months(DATE)</td>
<td>Real</td>
<td>Returns the time in months from the baseline date to the date represented by DATE, as a real number. This is an approximate figure, based on a month of 30.0 days. If DATE is before the baseline date, this function returns a negative number. You must include a valid date for the calculation to work appropriately. For example, you should not specify February 29, 2001, as the date. Because 2001 is a not a leap year, this date does not exist.</td>
</tr>
<tr>
<td>Function</td>
<td>Result</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>date_in_weeks(DATE)</code></td>
<td><code>Real</code></td>
<td>Returns the time in weeks from the baseline date to the date represented by <code>DATE</code>, as a real number. This is based on a week of 7.0 days. If <code>DATE</code> is before the baseline date, this function returns a negative number. You must include a valid date for the calculation to work appropriately. For example, you should not specify February 29, 2001 as the date. Because 2001 is not a leap year, this date does not exist.</td>
</tr>
<tr>
<td><code>date_in_years(DATE)</code></td>
<td><code>Real</code></td>
<td>Returns the time in years from the baseline date to the date represented by <code>DATE</code>, as a real number. This is an approximate figure based on a year of 365.0 days. If <code>DATE</code> is before the baseline date, this function returns a negative number. You must include a valid date for the calculation to work appropriately. For example, you should not specify February 29, 2001, as the date. Because 2001 is not a leap year, this date does not exist.</td>
</tr>
<tr>
<td><code>date_months_difference(DATE1, DATE2)</code></td>
<td><code>Real</code></td>
<td>Returns the time in months from <code>DATE1</code> to <code>DATE2</code>, as a real number. This is an approximate figure based on a month of 30.0 days. If <code>DATE2</code> is before <code>DATE1</code>, this function returns a negative number.</td>
</tr>
<tr>
<td><code>datetime_date(YEAR, MONTH, DAY)</code></td>
<td><code>Date</code></td>
<td>Creates a date value for the given <code>YEAR</code>, <code>MONTH</code>, and <code>DAY</code>. The arguments must be integers.</td>
</tr>
<tr>
<td><code>datetime_date(ITEM)</code></td>
<td><code>Date</code></td>
<td>Returns the date value for the given <code>ITEM</code> which may be a string, number, date, or timestamp. The function <code>datetime_date(STRING)</code> creates a date by parsing a string in the current date format. The <code>Date format</code> specified in the Stream Properties dialog box must be correct for this function to be successful. The function <code>datetime_date(NUMBER)</code> creates a date from a number, interpreted as a number of seconds since the base date (or epoch). Fractions of a day are truncated. The functions <code>datetime_date(DATE)</code> and <code>datetime_date(TIMESTAMP)</code> return a date unchanged, or the date part of a timestamp.</td>
</tr>
<tr>
<td><code>datetime_day(DATE)</code></td>
<td><code>Integer</code></td>
<td>Returns the day of the month from a given <code>DATE</code> or timestamp. The result is an integer in the range 1 to 31.</td>
</tr>
<tr>
<td><code>datetime_day_name(DAY)</code></td>
<td><code>String</code></td>
<td>Returns the full name of the given <code>DAY</code>. The argument must be an integer in the range 1 (Sunday) to 7 (Saturday).</td>
</tr>
<tr>
<td>Function</td>
<td>Result</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>datetime_hour(TIME)</td>
<td>Integer</td>
<td>Returns the hour from a TIME or timestamp. The result is an integer in the range 1 to 23.</td>
</tr>
<tr>
<td>datetime_in_seconds(DATETIME)</td>
<td>Real</td>
<td>Returns the number of seconds in a DATETIME.</td>
</tr>
<tr>
<td>datetime_minute(TIME)</td>
<td>Integer</td>
<td>Returns the minute from a TIME or timestamp. The result is an integer in the range 0 to 59.</td>
</tr>
<tr>
<td>datetime_month(DATE)</td>
<td>Integer</td>
<td>Returns the month from a DATE or timestamp. The result is an integer in the range 1 to 12.</td>
</tr>
<tr>
<td>datetime_month_name(MONTH)</td>
<td>String</td>
<td>Returns the full name of the given MONTH. The argument must be an integer in the range 1 to 12.</td>
</tr>
<tr>
<td>datetime_now</td>
<td>Timestamp</td>
<td>Returns the current time as a timestamp.</td>
</tr>
<tr>
<td>datetime_second(TIME)</td>
<td>Integer</td>
<td>Returns the second from a TIME or timestamp. The result is an integer in the range 0 to 59.</td>
</tr>
<tr>
<td>datetime_day_short_name(DAY)</td>
<td>String</td>
<td>Returns the abbreviated name of the given DAY. The argument must be an integer in the range 1 (Sunday) to 7 (Saturday).</td>
</tr>
<tr>
<td>datetime_month_short_name(MONTH)</td>
<td>String</td>
<td>Returns the abbreviated name of the given MONTH. The argument must be an integer in the range 1 to 12.</td>
</tr>
<tr>
<td>datetime_time(HOUR, MINUTE, SECOND)</td>
<td>Time</td>
<td>Returns the time value for the specified HOUR, MINUTE, and SECOND. The arguments must be integers.</td>
</tr>
<tr>
<td>datetime_time(ITEM)</td>
<td>Time</td>
<td>Returns the time value of the given ITEM.</td>
</tr>
<tr>
<td>datetime_timestamp(YEAR, MONTH, DAY, HOUR, MINUTE, SECOND)</td>
<td>Timestamp</td>
<td>Returns the timestamp value for the given YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.</td>
</tr>
<tr>
<td>datetime_timestamp(DATE, TIME)</td>
<td>Timestamp</td>
<td>Returns the timestamp value for the given DATE and TIME.</td>
</tr>
<tr>
<td>datetime_timestamp(NUMBER)</td>
<td>Timestamp</td>
<td>Returns the timestamp value of the given number of seconds.</td>
</tr>
<tr>
<td>datetime_weekday(DATE)</td>
<td>Integer</td>
<td>Returns the day of the week from the given DATE or timestamp.</td>
</tr>
<tr>
<td>datetime_year(DATE)</td>
<td>Integer</td>
<td>Returns the year from a DATE or timestamp. The result is an integer such as 2002.</td>
</tr>
<tr>
<td>date_weeks_difference(DATE1, DATE2)</td>
<td>Real</td>
<td>Returns the time in weeks from the date represented by DATE1 to the date represented by DATE2, as a real number. This is based on a week of 7.0 days. If DATE2 is before DATE1, this function returns a negative number.</td>
</tr>
<tr>
<td>Function</td>
<td>Result</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>date_years_difference</code> <code>(DATE1, DATE2)</code></td>
<td><code>Real</code></td>
<td>Returns the time in years from the date represented by <code>DATE1</code> to the date represented by <code>DATE2</code>, as a real number. This is an approximate figure based on a year of 365.0 days. If <code>DATE2</code> is before <code>DATE1</code>, this function returns a negative number.</td>
</tr>
<tr>
<td><code>time_before</code> <code>(TIME1, TIME2)</code></td>
<td><code>Boolean</code></td>
<td>Returns a value of true if <code>TIME1</code> represents a time before that represented by <code>TIME2</code>. Otherwise, this function returns a value of 0.</td>
</tr>
<tr>
<td><code>time_hours_difference</code> <code>(TIME1, TIME2)</code></td>
<td><code>Real</code></td>
<td>Returns the time difference in hours between the times represented by <code>TIME1</code> and <code>TIME2</code>, as a real number. If you select Rollover days/mins in the stream properties dialog box, a higher value of <code>TIME1</code> is taken to refer to the previous day. If you do not select the rollover option, a higher value of <code>TIME1</code> causes the returned value to be negative.</td>
</tr>
<tr>
<td><code>time_in_hours</code> <code>(TIME)</code></td>
<td><code>Real</code></td>
<td>Returns the time in hours represented by <code>TIME</code>, as a real number. For example, under time format <code>HHMM</code>, the expression <code>time_in_hours('0130')</code> evaluates to 1.5.</td>
</tr>
<tr>
<td><code>time_in_mins</code> <code>(TIME)</code></td>
<td><code>Real</code></td>
<td>Returns the time in minutes represented by <code>TIME</code>, as a real number.</td>
</tr>
<tr>
<td><code>time_in_secs</code> <code>(TIME)</code></td>
<td><code>Integer</code></td>
<td>Returns the time in seconds represented by <code>TIME</code>, as an integer.</td>
</tr>
<tr>
<td><code>time_mins_difference</code> <code>(TIME1, TIME2)</code></td>
<td><code>Real</code></td>
<td>Returns the time difference in minutes between the times represented by <code>TIME1</code> and <code>TIME2</code>, as a real number. If you select Rollover days/mins in the stream properties dialog box, a higher value of <code>TIME1</code> is taken to refer to the previous day (or the previous hour, if only minutes and seconds are specified in the current format). If you do not select the rollover option, a higher value of <code>TIME1</code> will cause the returned value to be negative.</td>
</tr>
<tr>
<td><code>time_secs_difference</code> <code>(TIME1, TIME2)</code></td>
<td><code>Integer</code></td>
<td>Returns the time difference in seconds between the times represented by <code>TIME1</code> and <code>TIME2</code>, as an integer. If you select Rollover days/mins in the stream properties dialog box, a higher value of <code>TIME1</code> is taken to refer to the previous day (or the previous hour, if only minutes and seconds are specified in the current format). If you do not select the rollover option, a higher value of <code>TIME1</code> causes the returned value to be negative.</td>
</tr>
</tbody>
</table>
Sequence Functions

For some operations, the sequence of events is important. The Clementine application allows you to work with the following record sequences:

- Sequences and time series
- Sequence functions
- Record indexing
- Averaging, summing, and comparing values
- Monitoring change—differentiation
- @SINCE
- Offset values
- Additional sequence facilities

For many applications, each record passing through a stream can be considered as an individual case, independent of all others. In such situations, the order of records is usually unimportant.

For some classes of problems, however, the record sequence is very important. These are typically time series situations, in which the sequence of records represents an ordered sequence of events or occurrences. Each record represents a snapshot at a particular instant in time; much of the richest information, however, might be contained not in instantaneous values but in the way in which such values are changing and behaving over time.

Of course, the relevant parameter may be something other than time. For example, the records could represent analyses performed at distances along a line, but the same principles would apply.

Sequence and special functions are immediately recognizable by the following characteristics:

- They are all prefixed by @.
- Their names are given in uppercase.

Sequence functions can refer to the record currently being processed by a node, the records that have already passed through a node, and even, in one case, records that have yet to pass through a node. Sequence functions can be mixed freely with other components of CLEM expressions, although some have restrictions on what can be used as their arguments.
You may find it useful to know how long it has been since a certain event occurred or a condition was true. Use the function @SINCE to do this—for example:

@SINCE(Income > Outgoings)

This function returns the offset of the last record where this condition was true—that is, the number of records before this one in which the condition was true. If the condition has never been true, @SINCE returns @INDEX +.

Sometimes you may want to refer to a value of the current record in the expression used by @SINCE. You can do this using the function @THIS, which specifies that a field name always applies to the current record. To find the offset of the last record that had a Concentration field value more than twice that of the current record, you could use:

@SINCE(Concentration > 2 * @THIS(Concentration))

In some cases the condition given to @SINCE is true of the current record by definition—for example:

@SINCE(ID = = @THIS(ID))

For this reason, @SINCE does not evaluate its condition for the current record. Use a similar function, @SINCE0, if you want to evaluate the condition for the current record as well as previous ones; if the condition is true in the current record, @SINCE0 returns 0.

### Available Sequence Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@MEAN(FIELD)</td>
<td>Real</td>
<td>Returns the mean average of values for the specified FIELD or FIELDS.</td>
</tr>
<tr>
<td>Function</td>
<td>Result</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>@MEAN(FIELD, EXPR)</td>
<td>Real</td>
<td>Returns the mean average of values for FIELD over the last EXPR records received by the current node, including the current record. FIELD must be the name of a numeric field. EXPR may be any expression evaluating to an integer greater than 0. If EXPR is omitted, or if it exceeds the number of records received so far, the average over all of the records received so far is returned.</td>
</tr>
<tr>
<td>@MEAN(FIELD, EXPR, INT)</td>
<td>Real</td>
<td>Returns the mean average of values for FIELD over the last EXPR records received by the current node, including the current record. FIELD must be the name of a numeric field. EXPR may be any expression evaluating to an integer greater than 0. If EXPR is omitted, or if it exceeds the number of records received so far, the average over all of the records received so far is returned. INT specifies the maximum number of values to look back. This is far more efficient than using just two arguments.</td>
</tr>
<tr>
<td>@DIFF1(FIELD)</td>
<td>Real</td>
<td>Returns the first differential of FIELD1. The single-argument form thus simply returns the difference between the current value and the previous value of the field. Returns 0 if the relevant previous records do not exist.</td>
</tr>
<tr>
<td>@DIFF1(FIELD1, FIELD2)</td>
<td>Real</td>
<td>The two-argument form gives the first differential of FIELD1 with respect to FIELD2. Returns 0 if the relevant previous records do not exist.</td>
</tr>
<tr>
<td>@DIFF2(FIELD)</td>
<td>Real</td>
<td>Returns the second differential of FIELD1. The single-argument form thus simply returns the difference between the current value and the previous value of the field. Returns 0 if the relevant previous records do not exist.</td>
</tr>
<tr>
<td>@DIFF2(FIELD1, FIELD2)</td>
<td>Real</td>
<td>The two-argument form gives the first differential of FIELD1 with respect to FIELD2. Returns 0 if the relevant previous records do not exist.</td>
</tr>
<tr>
<td>@INDEX</td>
<td>Integer</td>
<td>Returns the index of the current record. Indices are allocated to records as they arrive at the current node. The first record is given index 1, and the index is incremented by 1 for each subsequent record.</td>
</tr>
<tr>
<td>Function</td>
<td>Result</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>@LAST_NON_BLANK(FIELD)</td>
<td>Any</td>
<td>Returns the last value for FIELD that was not blank, according to any blank definition for FIELD in a Type node upstream of the current node, or satisfying the Blank If value of the current node, if this is a Filler node. If there are no nonblank values for FIELD in the records read so far, $null$ is returned.</td>
</tr>
<tr>
<td>@MAX(FIELD)</td>
<td>Number</td>
<td>Returns the maximum value for the specified italic.</td>
</tr>
<tr>
<td>@MAX(FIELD, EXPR)</td>
<td>Number</td>
<td>Returns the maximum value for FIELD over the last EXPR records received so far, including the current record. FIELD must be the name of a numeric field. EXPR may be any expression evaluating to an integer greater than 0.</td>
</tr>
<tr>
<td>@MAX(FIELD, EXPR, INT)</td>
<td>Number</td>
<td>Returns the maximum value for FIELD over the last EXPR records received so far, including the current record. FIELD must be the name of a numeric field. EXPR may be any expression evaluating to an integer greater than 0. INT specifies the maximum number of values to look back. This is far more efficient than using just two arguments.</td>
</tr>
<tr>
<td>@MIN(FIELD)</td>
<td>Number</td>
<td>Returns the minimum value for the specified FIELD.</td>
</tr>
<tr>
<td>@MIN(FIELD, EXPR)</td>
<td>Number</td>
<td>Returns the minimum value for FIELD over the last EXPR records received so far, including the current record. FIELD must be the name of a numeric field. EXPR may be any expression evaluating to an integer greater than 0.</td>
</tr>
<tr>
<td>@MIN(FIELD, EXPR, INT)</td>
<td>Number</td>
<td>Returns the minimum value for FIELD over the last EXPR records received so far, including the current record. FIELD must be the name of a numeric field. EXPR may be any expression evaluating to an integer greater than 0. INT specifies the maximum number of values to look back. This is far more efficient than using just two arguments.</td>
</tr>
</tbody>
</table>
### Appendix A

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@OFFSET(FIELD, EXPR)</td>
<td>Any</td>
<td>Retrieves values for a given field in previous or following records. It returns the value of the field named <code>FIELD</code> in the record offset from the current record by the value of <code>EXPR</code>. If <code>EXPR</code> is a (literal) integer, it may be positive or negative; a positive offset refers to a record that has already passed, while a negative one specifies a “lookahead” to a record that has yet to arrive. <code>EXPR</code> may also be an arbitrary CLEM expression, which is evaluated for the current record to give the offset. If this expression returns anything other than a non-negative integer, this causes an error—that is, it is not legal to have calculated lookahead offsets. For example, @OFFSET(Status, 1) returns the value of the Status field in the previous record.</td>
</tr>
<tr>
<td>@OFFSET(FIELD, EXPR, INT)</td>
<td>Any</td>
<td>Performs the same operation as the @OFFSET function with the addition of a third argument, <code>INT</code>, which specifies the maximum number of values to look back. This is far more efficient than using just two arguments. For example, @OFFSET(Status, –4) “looks ahead” four records in the sequence (that is, to records that have not yet passed through this node) to obtain the value. For lookahead (negative offset), the second argument must be a literal integer, not an expression. For positive offsets, though, any expression can be used.</td>
</tr>
<tr>
<td>@SDEV(FIELD)</td>
<td>Real</td>
<td>Returns the standard deviation of values for the specified <code>FIELD</code> or <code>FIELDS</code>.</td>
</tr>
<tr>
<td>@SDEV(FIELD, EXPR)</td>
<td>Real</td>
<td>Returns the standard deviation of values for <code>FIELD</code> over the last <code>EXPR</code> records received by the current node, including the current record. <code>FIELD</code> must be the name of a numeric field. <code>EXPR</code> may be any expression evaluating to an integer greater than 0. If <code>EXPR</code> is omitted, or if it exceeds the number of records received so far, the standard deviation over all of the records received so far is returned.</td>
</tr>
<tr>
<td>Function</td>
<td>Result</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>@SDEV(FIELD, EXPR, INT)</td>
<td>Real</td>
<td>Returns the standard deviation of values for FIELD over the last EXPR records received by the current node, including the current record. FIELD must be the name of a numeric field. EXPR may be any expression evaluating to an integer greater than 0. If EXPR is omitted, or if it exceeds the number of records received so far, the standard deviation over all of the records received so far is returned. INT specifies the maximum number of values to look back. This is far more efficient than using just two arguments.</td>
</tr>
<tr>
<td>@SINCE(EXPR)</td>
<td>Any</td>
<td>Returns the number of records that have passed since EXPR, an arbitrary CLEM expression, was true.</td>
</tr>
<tr>
<td>@SINCE(EXPR, INT)</td>
<td>Any</td>
<td>Adding the second argument, INT specifies the maximum number of records to look back.</td>
</tr>
<tr>
<td>@SINCE0(EXPR)</td>
<td>Any</td>
<td>Considers the current record, while @SINCE does not; @SINCE0 returns 0 if EXPR is true for the current record. If EXPR has never been true, INT is @INDEX+1</td>
</tr>
<tr>
<td>@SINCE0(EXPR, INT)</td>
<td>Any</td>
<td>Adding the second argument, INT specifies the maximum number of records to look back.</td>
</tr>
<tr>
<td>@SUM(FIELD)</td>
<td>Number</td>
<td>Returns the sum of values for the specified FIELD or FIELDS.</td>
</tr>
<tr>
<td>@SUM(FIELD, EXPR)</td>
<td>Number</td>
<td>Returns the sum of values for FIELD over the last EXPR records received by the current node, including the current record. FIELD must be the name of a numeric field. EXPR may be any expression evaluating to an integer greater than 0. If EXPR is omitted, or if it exceeds the number of records received so far, the sum over all of the records received so far is returned.</td>
</tr>
<tr>
<td>@SUM(FIELD, EXPR, INT)</td>
<td>Number</td>
<td>Returns the sum of values for FIELD over the last EXPR records received by the current node, including the current record. FIELD must be the name of a numeric field. EXPR may be any expression evaluating to an integer greater than 0. If EXPR is omitted, or if it exceeds the number of records received so far, the sum over all of the records received so far is returned. INT specifies the maximum number of values to look back. This is far more efficient than using just two arguments.</td>
</tr>
<tr>
<td>@THIS(FIELD)</td>
<td>Any</td>
<td>Returns the value of the field named FIELD in the current record. Used only in @SINCE expressions.</td>
</tr>
</tbody>
</table>
Appendix A

Global Functions

The functions @MEAN, @SUM, @MIN, @MAX, and @SDEV work on, at most, all of the records read up to and including the current one. In some cases, however, it is useful to be able to work out how values in the current record compare with values seen in the entire data set. Using a Set Globals node to generate values across the entire data set, you can access these values in a CLEM expression using the global functions.

For example:

@GLOBAL_MAX(Age)

returns the highest value of Age in the data set, while the expression

(Value - @GLOBAL_MEAN(Value)) / @GLOBAL_SDEV(Value)

expresses the difference between this record's Value and the global mean as a number of standard deviations. You can use global values only after they have been calculated by a Set Globals node. All current global values can be canceled by clicking the Clear Global Values button on the Globals tab in the Stream Properties dialog box.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@GLOBAL_MAX(FIELD)</td>
<td>Number</td>
<td>Returns the maximum value for FIELD over the whole data set, as previously generated by a Set Globals node. FIELD must be the name of a numeric field. If the corresponding global value has not been set, an error occurs.</td>
</tr>
<tr>
<td>@GLOBAL_MIN(FIELD)</td>
<td>Number</td>
<td>Returns the minimum value for FIELD over the whole data set, as previously generated by a Set Globals node. FIELD must be the name of a numeric field. If the corresponding global value has not been set, an error occurs.</td>
</tr>
<tr>
<td>@GLOBAL_SDEV(FIELD)</td>
<td>Number</td>
<td>Returns the standard deviation of values for FIELD over the whole data set, as previously generated by a Set Globals node. FIELD must be the name of a numeric field. If the corresponding global value has not been set, an error occurs.</td>
</tr>
</tbody>
</table>
### Functions Handling Blanks and Null Values

Using CLEM, you can specify that certain values in a field are to be regarded as “blanks,” or missing values. The following functions work with blanks:

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@BLANK(FIELD)</td>
<td>Boolean</td>
<td>Returns true for all records whose values are blank according to the blank handling rules set in an upstream Type node or Source node (Types tab).</td>
</tr>
<tr>
<td>@LAST_NON_BLANK(FIELD)</td>
<td>Any</td>
<td>Returns the last value for FIELD that was not blank, according to any blank definition for FIELD in a Type node upstream of the current node, or satisfying the Blank If value of the current node, if this is a Filler node. If there are no nonblank values for FIELD in the records read so far, $\text{null}$ is returned.</td>
</tr>
<tr>
<td>@NULL(FIELD)</td>
<td>Boolean</td>
<td>Returns true if the value of FIELD is the system-missing $\text{null}$. Returns false for all other values, including user-defined blanks. If you want to check for both, use @BLANK(FIELD) and @NULL(FIELD).</td>
</tr>
<tr>
<td>undef</td>
<td>Any</td>
<td>Used generally in CLEM to enter a $\text{null}$ value—for example, to fill blank values with nulls in the Filler node.</td>
</tr>
</tbody>
</table>

Blank fields may be “filled in” with the Filler node. In both Filler and Derive nodes (multiple mode only), the special CLEM function @FIELD refers to the current field(s) being examined.
Special Fields

Special functions are used to denote the specific fields under examination. For example, when deriving multiple fields at once, you should use `@FIELD` to denote “perform this derive action on the selected fields.” Using the expression $\log(\text{@FIELD})$ derives a new log field for each selected field.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@FIELD</td>
<td>Any</td>
<td>Performs an action on all fields specified in the expression context.</td>
</tr>
<tr>
<td>@TARGET</td>
<td>Any</td>
<td>When a CLEM expression is used in a user-defined analysis function, <code>@TARGET</code> represents the target field or “correct value” for the target/predicted pair being analyzed. This function is commonly used in an Analysis node.</td>
</tr>
<tr>
<td>@PREDICTED</td>
<td>Any</td>
<td>When a CLEM expression is used in a user-defined analysis function, <code>@PREDICTED</code> represents the predicted value for the target/predicted pair being analyzed. This function is commonly used in an Analysis node.</td>
</tr>
</tbody>
</table>

Obsolete Features and Functions

The following functions, used in version 6.0 and earlier, are no longer supported in Clementine:

<table>
<thead>
<tr>
<th>Old Functions</th>
<th>New Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>is_number</td>
</tr>
<tr>
<td>integer</td>
<td>is_integer</td>
</tr>
<tr>
<td>real</td>
<td>is_real</td>
</tr>
<tr>
<td>string</td>
<td>is_string</td>
</tr>
<tr>
<td>strnumber</td>
<td>to_number</td>
</tr>
<tr>
<td>strinteger</td>
<td>to_integer</td>
</tr>
<tr>
<td>strfloat</td>
<td>to_real</td>
</tr>
<tr>
<td>@AVE</td>
<td>@MEAN</td>
</tr>
<tr>
<td>@GLOBAL_AVE</td>
<td>@GLOBAL_MEAN</td>
</tr>
</tbody>
</table>
Scripting Language Reference

Scripting Reference Overview

You can use statements in the Clementine scripting language to perform the following tasks:

- Execute nodes.
- Set options for individual nodes.
- Manipulate nodes, SuperNodes, and output.
- Manipulate generated models.
- Load and save states and streams.

The Clementine scripting language consists of:

- A set of scripting statements
- A format for referring to nodes
- A scripting expression language used for the values of parameters and node properties
- A format for expressing lists of constants

These functions and components of scripting in Clementine are discussed throughout this section.

Scripting Syntax

To improve clarity during parsing, the following rules should be followed when working with scripts in Clementine:

- Variable names, such as income or refererID, must be unquoted.
Appendix B

- Global parameter references, such as 'SP-Maxvalue', should be single-quoted.
- File names, such as "druglearn.str", should be double-quoted.
- Parameter references, such as ^mystream, should be preceded with a ^ symbol.
- Node names, such as databasenode or Na_to_K, can be unquoted or single-quoted. Note: Names must be quoted if they include spaces or special characters. You cannot, however, use a node name in a script if the name starts with a number, such as '2a_referrerID'.
- String literals, such as "Web graph of BP and Drug" or "High", should be double-quoted or single-quoted if the context prohibits the use of object references.
- CLEM expressions, such as "Age >= 55", should be double-quoted.
- If you use quotation marks within a CLEM expression, make sure that each quotation mark is preceded by a backslash (\)—for example:
  set node.parameter=" BP="HIGH\"".

Note: Scripts written for previous versions of Clementine will continue to work as they did before; however, it is recommended that you use the above guidelines for improved clarity. The script checker available in all scripting dialog boxes will flag ambiguous syntax.

Inserting Comments and Continuations

The following characters are used in scripting to denote comments and continuations.

<table>
<thead>
<tr>
<th>Character</th>
<th>Usage</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>The hash sign is a comment. The rest of the line is ignored.</td>
<td>#This is a single-line comment.</td>
</tr>
<tr>
<td>/</td>
<td>A line ending with a slash indicates that the statement continues on the next line.</td>
<td>See example below.</td>
</tr>
<tr>
<td>/*</td>
<td>The sequence /* indicates the beginning of a comment. Everything is ignored until a */ end comment marker is found.</td>
<td>See example below.</td>
</tr>
</tbody>
</table>

Following are examples of multiline comments and continuations:

/* This is a
multi line
comment
*/

set :fixedfilenode.fields = ["Age" 1 3]/
{"Sex" 5 7} {"BP" 9 10} {"Cholesterol" 12 22}/
{"Na" 24 25} {"K" 27 27} {"Drug" 29 32]}

Operators in Scripts

In addition to the usual CLEM operators, you can manipulate list-type local or slot parameters in scripts using the “+” and “−” operators. The “+” adds an element to the list, and the “−” operator removes an item. Here is an example:

```
var z  # create a new local parameter
set z = [1 2 3]  # set it to the list containing 1, 2, and 3
set z = z + 4  # add an element; z now equals [1 2 3 4]
```

These operators cannot be used with Stream, SuperNode, or Session parameters, nor outside of scripts in general CLEM expressions (such as a formula in a Derive node).

Guidelines for Referring to Nodes in Scripts

There are several node-specific guidelines to follow for correct scripting syntax.

- You can specify nodes by name—for example, DRUG1n. You can qualify the name by type—for example, Drug:neuralnetnode refers to a Neural Net node called Drug and not to any other kind of node.
- You can specify nodes by type only. For example, :neuralnetnode refers to all Neural Net nodes. This statement is not restricted to nodes of any particular name.
- Node types are words—for example, samplenode, neuralnetnode, and kmeansnode. Although you can omit the suffix node, it is recommended that you include it because it makes identifying errors in scripts easier.
- You can supply node names and types as the values of parameters by using the ^ syntax. For example, where a node name is required, ^n means the node whose name is stored in the parameter n, and Drug:^t means the node called Drug, whose type is stored in the parameter t.
The same rules apply to generated model nodes. You can use the name of the node on the generated models palette in the managers window when specifying a generated model in scripting. For more information, see “Manipulating the Generated Models Palette” on page 722.

**Setting Properties and Parameters**

Using scripting, you can specify the value of node properties as well as expressions for local and global parameters. The following command is used to set the value of the parameter (or node property, also referred to as a slot parameter):

```
set PARAMETER = EXPRESSION
```

PARAMETER can be:

- A global parameter or variable, such as `x`
- A local parameter or variable, such as `my_node`
- A special variable, such as `stream` where `stream` is the current stream
- A node property, such as `Na_to_K:derivenode.formula_expr`
- A script command, such as `save stream`

EXPRESSION can be:

- A CLEM expression valid in scripting, such as "Age >= 55"
- A script command that returns a value, such as `load`, `create`, or `get`
- A literal value, such as 1 or `Include`

### Examples

Following are examples of set expressions used to specify parameter values, node properties, and CLEM expressions used in node properties:

```plaintext
set p = 1
set minvalue = 21
set :derivenode.new_name = "Ratio of Na to K"
set :derivenode.formula_expr = "Na / K"
set my_node = get node :plotnode
```
Multiple Expressions

You can assign multiple expressions to properties for nodes (also called slot parameters) in a single operation. It is used when multiple changes need to be made to a node before the data model is determined. The format used to set multiple properties is:

```
set NODE {
    NODEPROPERTY1 = EXPRESSION1
    NODEPROPERTY2 = EXPRESSION2
}
```

For example, suppose you want to set multiple properties for a Sample node. To do so, you could use the following multiset command:

```
set :samplenode {
    max_size = 200
    mode = "Include"
    sample_type = "First"
}
```

Parameters in Scripting

The scripting language often uses parameters to refer to variables in the current script or at a variety of levels within Clementine.

- **Local parameters** refer to variables set for the current script using the `var` command.
- **Global parameters** refer to Clementine parameters set for streams, SuperNodes, and sessions.

These types of parameters are discussed further in the following topics.

Local Parameters

Local parameters are parameters set locally for access to objects and values of any type by the current script only. Local parameters are also referred to as **local variables** and are declared with the `var` command. Using the `var` command for local
parameters helps maintain the distinction between local parameters (variables) and **global parameters**, which can be set for a session, stream, or SuperNode and can contain strings or numbers only.

When referring to local parameters in scripting statements, be sure to use the `^` symbol preceding the parameter name. For example, the following script is used to set a local parameter and then refers to that parameter:

```plaintext
var my_node
set my_node = create distributionnode
rename ^my_node as "Distribution of Flag"
```

When resolving variable references, the local parameter list is searched before the global parameter list. For example, if a variable `x` existed as a local parameter and a global parameter, using the syntax `$P-X` in a scripting statement would ensure that the global parameter variable is used rather than the local one.

## Global Parameters

When you use ordinary parameters such as stream, session, or SuperNode parameters in a script, these parameters are called **global parameters**. Global parameters are often used in scripting as part of a CLEM expression in which the parameter value is specified in the script.

## Setting Parameters

You can set parameters using the `set` command and the following syntax:

```plaintext
set foodtype = pizza
```

If there are no nodes or existing parameters named `foodtype`, this command creates a parameter called `foodtype` with a default value of `pizza`.

- The parameter is created for a stream if the command is part of a stream script, or a SuperNode if the script is a SuperNode script.
- If the command is used as a startup flag on the command line or a standalone script, the parameter becomes a session parameter.
Referring to Parameters

You can refer to previously created parameters by encapsulating them in single quotes, prefaced with the string $P—for example, $P-minvalue'. You can also refer simply to the parameter name, such as minvalue. The value for a global parameter is always a string or number. For example, you can refer to the foodtype parameter and set a new value using the following syntax:

set foodtype = beer

You can also refer to parameters within the context of a CLEM expression used in a script. As an example, the following script sets the properties for a Select node to include records where the value for Age is greater than that specified by the stream parameter called cutoff. The parameter is used in a CLEM expression with the proper syntax for CLEM—'$P-cutoff':

```plaintext
set :selectnode {
    mode = "Include"
    condition = "Age >= '$P-cutoff'"
}
```

The script above uses the default value for the stream parameter called cutoff. You can specify a new parameter value by adding the following line to the script above the Select node specifications:

```plaintext
set cutoff = 50
```

The resulting script selects all records where the value of Age is greater than 50.

For more information, see “Parameters” in Appendix A on page 669.
Using CLEM in Scripts

You can use CLEM expressions, functions, and operators within scripts used in Clementine; however, your scripting expression cannot contain calls to any @ functions, date/time functions, and bitwise operations. Additionally, the following rules apply to CLEM expressions in scripting:

- Parameters must be specified in single quotes and with the $P- prefix.
- CLEM expressions must be encased in quotes. If the CLEM expression itself contains quoted strings or quoted field names, the embedded quotes must be preceded by a backslash( \\). For more information, see “Scripting Syntax” on page 697.

You can use global values such as GLOBAL_MEAN(Age) in scripting, however you cannot use the @GLOBAL function itself within the scripting environment.

Examples of CLEM expressions used in scripting are:

```plaintext
set :balancenode.directives = [{1.3 "Age > 60"]
set :fillernode.condition = "(Age > 60) and (BP = "High")"
set :derivenode.formula_expr = "substring(5, 1, Drug)"
set Flag:derivenode.flag_expr = "Drug = X"
set :selectnode.condition = "Age >= $P-cutoff"
set :derivenode.formula_expr = "Age - GLOBAL_MEAN(Age)"
```

Creating Nodes and Streams

The commands below are used to create a node or stream of the given specification and modifiers. The modifiers are specific to the type of object being created.

Node Creation

create NODE NODE_POSITION

In addition to specifying the creation of a node, you can also specify position and connection options, as follows:

NODE_POSITION
You can also create a node using variables to avoid ambiguity. For instance, in the example below, a Type node is created and the reference variable \( x \) is set to contain a reference to that Type node. You can then use the variable \( x \) to return the object referenced by \( x \) (in this case, the Type node) and perform additional operations, such as renaming, positioning, or connecting the new node.

```plaintext
var x
set x = create typenode
rename \(^x\) as "mytypenode"
position \(^x\) at 200 200
var y
set y = create varfilenode
rename \(^y\) as "mydatasource"
position \(^y\) at 100 200
connect \(^y\) to \(^x\)
```

The above example creates two nodes, renames each, positions them, and finally connects them on the stream canvas.

**Figure B-1**

*Nodes created using variables*

You can also use the reserved word—`node`—as a special variable in circumstances similar to the one above. For instance, you might use a script, such as the following when creating a stream:

```plaintext
set node = create typenode
rename \(^\text{node}\) as "mytypenode"
position \(^\text{node}\) at 200 200
set node = create varfilenode
rename \(^\text{node}\) as "mydatasource"
position \(^\text{node}\) at 100 200
```
connect mydatasource to mytypenode

The script above is a good illustration of how to avoid ambiguity in scripting. The variable node is used to refer to specific objects and rename them unambiguously. At the end of the example script above, the unambiguous node names are used to connect the nodes on the stream canvas.

*Note:* Special variables, such as node, can be re-used to reference multiple nodes. Simply use the set command to reset the object referenced by the variable. For more information, see “Setting the Current Object” on page 709.

**Stream Creation**

create STREAM DEFAULT_FILENAME

This creates a new stream with either the default stream name or a name of your choice. The newly created stream is returned as an object.

**X-Y Positioning**

Positioning nodes on the stream canvas uses an invisible x-y grid. You can use the image below as a reference for the x-y grid coordinates.
Loading and Saving Objects

Opening Streams

You can open a stream by specifying the filename and location of the file.

Open stream FILENAME

This returns the stream loaded from the file.
**Loading Objects**

To open a variety of objects, use the following command:

`load OBJECT_TYPE FILENAME`

This returns the object loaded from the file.

Object types are:
- `stream`
- `project`
- `node`
- `model`
- `generated palette`
- `state`

**Saving Objects**

You can also save an object to a file. The first format is valid for streams and projects only, and uses the object’s default name if there is one or generates an error if there is not.

`save OBJECT`
`save OBJECT as FILENAME`

Objects are:
- An object reference, such as `node NODE model MODEL`
- A variable containing an object that can be saved
- Special variables such as `stream` (the current stream) or `generated palette` (the generated models tab/palette)


Retrieving Objects

The following commands retrieve an existing object of the given type. For general objects, retrieval is on the basis of the object name. For nodes, retrieval is on the basis of a node description. An error is raised if the specified object cannot be found.

get OBJECT_TYPE NAME
get node NODE

This returns the object retrieved.

Object types are:
- stream
- node

Setting the Current Object

You can refer to the “current” object in scripting using predefined, special variables. The words listed below are reserved in Clementine scripting to indicate the “current” object and are called special variables:
- node—the current node
- stream—the current stream
- model—the current model
- generated palette—the generated models palette on the Models tab of the managers window
- output—the current output
- project—the current project

Each of these special variables can be used in a scripting statement by assigning a variable of the correct type. For example, in the following statement, stream is a special variable referring to the current stream:

save stream as "C:/My Streams/Churn.str"
Note: Throughout this guide, the presence of “current” objects, such as streams or nodes is noted for each scripting statement that returns an object as the “current” object. For example, when opening and loading a stream, the stream specified becomes the current stream.

**Using Special Variables**

An arbitrary object, such as a stream, can be made the current object by assigning that object to the corresponding special variable. For example, making a stream the current stream can be done by assigning a stream to the special variable `stream`—for example:

```plaintext
set stream = my_stream
```

Assigning a value of the wrong type to a special variable causes a run-time error. In cases where the special variable can be used, any variable can also be used. For example, saving the current stream can be carried out with:

```plaintext
save stream as 'C:/My Streams/Churn.str'
```

It is also valid to say:

```plaintext
save my_stream as 'C:/My Streams/Churn.str'
```

where `my_stream` has previously been assigned a stream value.

**Closing and Deleting Objects**

Once you have created or loaded an object, you can use several commands to close or delete it.

**Closing Streams**

```plaintext
close STREAM
```

The above command closes the specified `STREAM`, but does not close Clementine or any other streams.
**Deleting Nodes**

```
delete NODE
```

The above command deletes the specified node from the current stream in the stream canvas. NODE can be a standard node or a generated model node in the stream canvas.

**Manipulating Streams**

In addition to the properties common to all objects, streams include a number of properties that can be used to change the behavior of the stream through scripting.

**Stream Properties**

Using scripting, you can specify a number of properties for streams. For example, using the special stream variable, `^stream`, you can set the following types of properties.

```
set ^stream.execute_method = "Script"
set ^stream.date_format = "MM/DD/YY"
```

**Stream Execution**

You can execute streams using the following statements.

```
execute NODE
```

The above command executes the section of the current stream that includes the specified node. If the node is a terminal node, then this executes the stream section terminated by the node. If the node is not a terminal node, then execution is equivalent to the Execute From Here pop-up menu option.

```
execute NODE N
```

The above command executes the section of the current stream that includes the specified node. The search for the node begins at the Nth node. Different values of N are guaranteed to execute different nodes.
execute_all
The above command executes all terminal nodes in the current stream.

execute_script
The above command executes the stream script associated with the current stream.

*Note:* Scripts associated with different streams can be executed by setting the stream as the current stream or by using the with command.

**Reassigning the Current Stream**

```plaintext
with stream STREAM
  STATEMENT(s)
endwith
```

This syntax is used to temporarily reassign the current stream to be the specified STREAM—for example:

```plaintext
with stream STREAM
  create typenode
  execute_script
endwith
```

The above statements execute the create action and execute the stream’s script with the specified STREAM set as the current stream. The original current stream is restored once each statement has been executed.

Conditional statements and loop constructs can also be included—for example:

```plaintext
with stream STREAM
  for I from 1 to 5
    set :selectnode.expression = 'field > ' >< (I * 10)
  execute
  endfor
endwith
```
This will set the current stream to \textit{STREAM} for all expressions within the loop and restore the original value when the loop has completed.

\textbf{Closing Streams}

\begin{verbatim}
close STREAM
\end{verbatim}

The above syntax closes the specified \textit{STREAM}.

\textbf{Node Manipulation}

In addition to the standard properties for objects such as creating, saving, and loading, there are numerous node-specific properties that can be used to change the behavior of the node. These properties, as well as general guidelines for manipulating and referring to nodes in scripts are discussed in the following topics.

\textbf{Node Names in Scripting}

Once created, nodes in scripts should be referred to using the form:

\begin{verbatim}
NAME:TYPE
\end{verbatim}

\textit{NAME} is the name of a node, and type is its type. At a minimum, you must include either \textit{NAME} or \textit{TYPE}. You may omit one, but you may not omit both. For example, the following command creates a new Derive node (new nodes do not use the colon) between an existing Variable File node called \textit{drug1n} and an existing Plot node.

\begin{verbatim}
create derivenode connected between drug1n and :plotnode
\end{verbatim}

You can also precede either \textit{NAME} or \textit{TYPE} by a ^ symbol to indicate the presence of a parameter. For example, ^\textit{PARAMETER} means that the relevant component (the name or type of the node) is the value of the parameter \textit{PARAMETER}. This ^ notation may also be used when supplying generated model names and node property (slot parameter) names or types—for example:

\begin{verbatim}
Drug:^\textit{t}
\end{verbatim}
means a node named \textit{Drug} where \( t \) is a parameter for the type of node with a value of \texttt{c50node}. Essentially, the above reference can be translated as:

\texttt{Drug:c50node}

Similarly, a parameter can be used for the node name. For example, the following node references:

\texttt{^n:derivenode}
\texttt{^n}

can both be used in a context where a node name is required, and where \( n \) has the value \texttt{Na_to_K}. This refers to the node named \texttt{Na_to_K}.

The node type specifies the type of the node and can be any of the nodes described in the following table or the name of any node defined by the Clementine External Module Interface. The suffix \texttt{node} may be omitted. Although you may omit the suffix \texttt{node}, it is recommended that you include it because it makes identifying errors in scripts easier. Where the node name or name/type combination is ambiguous—that is, where it could refer to more than one node—an error is raised.

\textit{Node Names Reference}

The following table contains a complete list of node names used for scripting.

\textbf{Table B-1}
\textit{Node names for scripting}

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Node Name in User Interface</th>
<th>Scripting Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sources</td>
<td>Var. File</td>
<td>\texttt{variablefilenode}</td>
</tr>
<tr>
<td></td>
<td>Fixed File</td>
<td>\texttt{fixedfilenode}</td>
</tr>
<tr>
<td></td>
<td>Database</td>
<td>\texttt{databasenode}</td>
</tr>
<tr>
<td></td>
<td>SAS Import</td>
<td>\texttt{sasimportnode}</td>
</tr>
<tr>
<td></td>
<td>SPSS Import</td>
<td>\texttt{spssimportnode}</td>
</tr>
<tr>
<td></td>
<td>User Input</td>
<td>\texttt{userinputnode}</td>
</tr>
<tr>
<td><strong>Node Type</strong></td>
<td><strong>Node Name in User Interface</strong></td>
<td><strong>Scripting Syntax</strong></td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Record Operations</td>
<td>Sample</td>
<td>samplenode</td>
</tr>
<tr>
<td></td>
<td>Select</td>
<td>selectnode</td>
</tr>
<tr>
<td></td>
<td>Merge</td>
<td>mergenode</td>
</tr>
<tr>
<td></td>
<td>Balance</td>
<td>balancenode</td>
</tr>
<tr>
<td></td>
<td>Sort</td>
<td>sortnode</td>
</tr>
<tr>
<td></td>
<td>Aggregate</td>
<td>aggregatenode</td>
</tr>
<tr>
<td></td>
<td>Distinct</td>
<td>distinctnode</td>
</tr>
<tr>
<td></td>
<td>Append</td>
<td>appendnode</td>
</tr>
<tr>
<td>Field Operations</td>
<td>Filter</td>
<td>filternode</td>
</tr>
<tr>
<td></td>
<td>Type</td>
<td>typenode</td>
</tr>
<tr>
<td></td>
<td>Derive</td>
<td>derivenode</td>
</tr>
<tr>
<td></td>
<td>Filler</td>
<td>fillernode</td>
</tr>
<tr>
<td></td>
<td>SetToFlag</td>
<td>settoflagnode</td>
</tr>
<tr>
<td></td>
<td>History</td>
<td>historynode</td>
</tr>
<tr>
<td></td>
<td>Binning</td>
<td>binningnode</td>
</tr>
<tr>
<td></td>
<td>Reclassify</td>
<td>reclassifynode</td>
</tr>
<tr>
<td></td>
<td>Reorder</td>
<td>reordernode</td>
</tr>
<tr>
<td>Graphs</td>
<td>Plot</td>
<td>plotnode</td>
</tr>
<tr>
<td></td>
<td>Histogram</td>
<td>histogramnode</td>
</tr>
<tr>
<td></td>
<td>Distribution</td>
<td>distributionnode</td>
</tr>
<tr>
<td></td>
<td>Collection</td>
<td>collectionnode</td>
</tr>
<tr>
<td></td>
<td>Evaluation</td>
<td>evaluationnode</td>
</tr>
<tr>
<td></td>
<td>Web</td>
<td>webnode or directedwebnode</td>
</tr>
<tr>
<td></td>
<td>Multiplot</td>
<td>multiplotnode</td>
</tr>
</tbody>
</table>
### Node Type

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Node Name in User Interface</th>
<th>Scripting Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling</td>
<td>Neural Net</td>
<td>neuralnetnode</td>
</tr>
<tr>
<td></td>
<td>Build C5.0</td>
<td>c50node</td>
</tr>
<tr>
<td></td>
<td>Kohonen</td>
<td>kohonennode</td>
</tr>
<tr>
<td></td>
<td>Linear Reg.</td>
<td>regressionnode</td>
</tr>
<tr>
<td></td>
<td>Logistic Reg.</td>
<td>logregnode</td>
</tr>
<tr>
<td></td>
<td>C&amp;R Tree</td>
<td>cartnode</td>
</tr>
<tr>
<td></td>
<td>Factor/PCA</td>
<td>factornode</td>
</tr>
<tr>
<td></td>
<td>TwoStep</td>
<td>twostepnode</td>
</tr>
<tr>
<td></td>
<td>GRI</td>
<td>grinode</td>
</tr>
<tr>
<td></td>
<td>Apriori</td>
<td>apriorinode</td>
</tr>
<tr>
<td></td>
<td>Kmeans</td>
<td>kmeansnode</td>
</tr>
<tr>
<td></td>
<td>Sequence</td>
<td>sequencenode</td>
</tr>
<tr>
<td>Generated Models</td>
<td>Neural Net</td>
<td>applyneuralnetnode</td>
</tr>
<tr>
<td></td>
<td>Build C5.0</td>
<td>applyc50node</td>
</tr>
<tr>
<td></td>
<td>Kohonen</td>
<td>applykohonennode</td>
</tr>
<tr>
<td></td>
<td>Linear Reg.</td>
<td>applyregressionnode</td>
</tr>
<tr>
<td></td>
<td>Logistic Reg.</td>
<td>applylogregnode</td>
</tr>
<tr>
<td></td>
<td>C&amp;R Tree</td>
<td>applycartnode</td>
</tr>
<tr>
<td></td>
<td>Factor/PCA</td>
<td>applyfactornode</td>
</tr>
<tr>
<td></td>
<td>TwoStep</td>
<td>applytwostepnode</td>
</tr>
<tr>
<td></td>
<td>GRI</td>
<td>applygrinode</td>
</tr>
<tr>
<td></td>
<td>Apriori</td>
<td>applyapriorinode</td>
</tr>
<tr>
<td></td>
<td>Kmeans</td>
<td>applykmeansnode</td>
</tr>
<tr>
<td></td>
<td>Sequence</td>
<td>applysequencenode</td>
</tr>
<tr>
<td></td>
<td>Generated Rulesets</td>
<td>applyrulenode</td>
</tr>
</tbody>
</table>
### Node Type | Node Name in User Interface | Scripting Syntax
--- | --- | ---
Output | Table | tablenode
| Analysis | analysisnode
| Matrix | matrixnode
| Statistics | statisticsnode
| Set Globals | setglobalsnode
| Report | reportnode
| File | outputfilenode
| Database Output | databaseexportnode
| Quality | qualitynode
| SPSS Procedure | spssprocedurenodeline
| SAS Export | sasexportnode
| Publisher | publishernode
| SPSS Export | spssexportnode
| Excel | excelnode
| Data Audit | dataauditnode

#### Node Manipulation Commands

There are a number of commands used to manipulate nodes.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>Used to position the node in the stream canvas.</td>
</tr>
<tr>
<td>rename</td>
<td>Used to rename the specified node.</td>
</tr>
<tr>
<td>duplicate</td>
<td>Used to duplicate the specified node on the stream canvas.</td>
</tr>
<tr>
<td>delete</td>
<td>Used to remove all connections and delete the specified node.</td>
</tr>
<tr>
<td>flush</td>
<td>Used to flush the cache on a specified node. <code>flush_all</code> is also available to flush the cache on all nodes in the current stream.</td>
</tr>
</tbody>
</table>

These commands for manipulating nodes are discussed further below. Node-specific properties, also called **slot parameters**, are discussed in a separate topic. For more information, see “Node-Specific Properties” on page 721.
**Positioning Nodes**

There are a number of methods used to position nodes and models in the stream canvas.

position NODE NODE_POSITION

The above statement moves the specified node to the node position. This may also include a connection specification. **NODE_POSITION** can be specified using the following position/connection modifiers:

- at X Y
- between NODE1 and NODE2
- connected between NODE1 and NODE2

For example, to position an already created node, you can use any of the following methods:

position Drug:net at 100 100

This statement positions the neural net model called **Drug** at coordinates 100, 100.

position Drug:net between DRUG2n and analysis

This statement positions the net as precisely as possible between the two nodes named **DRUG2n** and **analysis**, ignoring any snap-to-grid settings.

position Drug:net connected between DRUG2n and analysis

This statement positions the net model between **Drug2n** and makes connections from **Drug2n** to the **net** and from the **net** to **analysis**, respectively.

**Renaming Nodes**

Once you have created a node, you can rename it using the following syntax:

rename NODE as NEWNAME
The above statement renames the specified node to the supplied name. For example, to rename a source node reflecting a new data source, you could use a command similar to the following:

rename :varfilenode as "testdata"

**Duplicating Nodes**

duplicate NODE as NEWNAME
duplicate NODE as NEWNAME NODE_POSITION

The statements above duplicates the specified node giving it the name provided. The second statement also allows the node to be positioned using the positional modifiers discussed above. This returns the newly created node.

**Flushing the Cache for a Node**

flush NODE

The above statement flushes the data cache of a node. If the cache is not enabled or is not full, this operation does nothing. Disabling the cache also flushes the cache.

flush_all

The above statement flushes the data caches of all nodes in the current stream.

**Deleting Nodes**

You can delete nodes using either of the following methods.

delete NODE

This statement deletes the specified node from the current stream. For this statement to function, the specified node must already exist—for example:

delete Na_to_K:derivenode

This statement deletes the Na_to_K node.
This statement deletes the last model inserted with the insert model statement. For this statement to function, both of the following conditions must be satisfied:

- The insert model statement must have been executed at least once within the current script execution.
- The node that the insert model statement created must still exist.

**Connecting and Disconnecting Nodes**

The following statements are used to connect and disconnect nodes in the stream canvas. For example, using object create properties in conjunction with connection statements, you can create a new Type node positioned between the two specified nodes:

```
create typednode connected between :databasenode and :filternode
```

**Available Commands**

connect NODE1 to NODE2

The above statement creates a connection from node 1 to node 2. For example, `connect :net to Analysis` will make a connection from the neural net model to a node called Analysis.

connect NODE1 between NODE2 and NODE3

The above statement creates a connection from node 2 to node 1 and from node 1 to node 3. This is a commonly used statement for quick stream building. For example, `create derivenode between drug1n and :selectnode` will add a Derive node and create connections between all three nodes if they are not already present.

disconnect NODE

The above statement deletes all connections to and from the node.

disconnect NODE1 from NODE2

The above statement deletes the connection from node 1 to node 2.

disconnect NODE1 between NODE2 and NODE3
The above statement deletes the connection from node 2 to node 1 and from node 1 to node 3.

Node-Specific Properties

There are many node-specific properties used to set options found in the user-interface dialog boxes for each node. These node properties are also referred to as slot parameters. For example, to create a stream and specify options for each node, you would use a script similar to this one:

```plaintext
create varfilenode at 100 100
set :varfilenode {
  full_filename = "demos/drug1n"
  read_field_names = "True"
}
create tablenode at 400 100
create samplenode connected between :varfilenode and :tablenode
set :samplenode {
  max_size = 200
  mode = "Include"
  sample_type = "First"
}
create plotnode at 300 300
create derivenode connected between drug1n and :plotnode
set :derivenode {
  new_name = "Ratio of Na to K"
  formula_expr = "Na / K"
}
set :plotnode {
  x_field = 'Ratio of Na to K'
  y_field = 'Age'
  color_field = 'BP'
}
```

The above script uses a combination of general and specific node properties to create a functional data stream. The multiset commands (contained within { }) are used to specify node-specific properties such as reading data files, CLEM expressions, and color overlay fields. For more information, see “Properties Reference Overview” in Appendix D on page 737.
Manipulating the Generated Models Palette

In addition to the standard properties used to manipulate objects in Clementine, there are a number of model-specific properties that you can use to work with the models in the generated models palette (also called the Models tab in the manager window).

This statement loads the complete contents of the specified generated models palette:

export generated MODEL in DIRECTORY

The next statement exports the specified model in the named directory. This statement exports C code for those nodes that support C code export. For any other exports, this statement generates an error. For this statement to work, the specified model must be present on the generated models palette and must be the only model with that name; also, the named directory must exist.

export_xml generated MODEL in DIRECTORY

Duplicate Model Names

When using scripts to manipulate generated models, you should be aware that allowing duplicate model names can result in script ambiguity. It's a good idea to require unique names for generated models when scripting.

To set options for duplicate model names:

- From the menus, choose:
  - Tools
  - User Options

- Click the Notifications tab.

- Select Replace previous model to restrict duplicate naming for generated models.

Adding Models to a Stream

There are several ways to add generated models to the current stream.
insert model MODEL
insert model MODEL NODE_POSITION

Both statements are used to insert the specified model from the generated models tab/palette into the stream. The second statement includes a positional specification.

For reference, NODE_POSITION modifiers for nodes are:
- at X Y
- between NODE1 and NODE2
- connected between NODE1 and NODE2

This returns the model added to the stream.

**Deleting Models**

delete last model

Deletes the last model inserted into the stream with the insert model statement. The insert model statement must have been executed at least once for the stream within the current script, and the node that was created must exist.

delete model MODEL

This deletes the named model from the generated models palette.

clear generated palette

This clears the generated models palette.

**Exporting Models**

Models can be exported to a specific directory with a specific filename using the following syntax:

export model MODEL in DIRECTORY format FORMAT
export model MODEL as FILENAME format FORMAT
The following formats are supported:

- pmml
- c_code
- modltext
- modelhtml

The options modelhtml and modeltext export the model tab of a generated model in either HTML or plain text. For example, if you have generated a model called assocapriori, you could export the model using the following command:

```export model 'assocapriori' as 'C:\temp\assoc_apriori' format modelhtml```

This creates an HTML file with the model tab results displayed in table format.

**Figure B-3**

Association model tab exported as HTML

![Association model tab](image)

### Manipulating SuperNodes

SuperNodes include a number of properties that can be used to change the behavior of the SuperNode.
### Property

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td></td>
<td>Provides access to the parameters specified within a SuperNode.</td>
</tr>
<tr>
<td>default_execute</td>
<td>SCRIPT or NORMAL</td>
<td>(Terminal SuperNodes only.) This property defines whether executing the terminal SuperNode simply executes each subterminal node or executes the SuperNode script.</td>
</tr>
</tbody>
</table>

For more information, see “SuperNode Properties” in Appendix D on page 794.

### Results Manipulation

Terminal nodes include a read-only parameter called output that can be used to access the most recently generated object. This release includes properties to allow script access to the attributes and values in the data that was generated in a Table node—for example:

```plaintext
set num_rows = :tablenode.output.row_count
set num_cols = :tablenode.output.column_count
```

Attempting to access the value of a generated object slot where the object has not been created will generate an error. The values within the data set underlying a particular generated object are accessible using the value command:

```plaintext
set table_data = :tablenode.output
set last_value = value table_data at num_rows num_cols
```

Indexing is from 1.

The following properties are common to all result objects:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_count</td>
<td>Returns the number of rows in the data.</td>
</tr>
<tr>
<td>column_count</td>
<td>Returns the number of columns in the data.</td>
</tr>
</tbody>
</table>
Accessing Data

The following statement is used to access data from the results object.

\[
\text{value RESULT at ROW COLUMN}
\]

This returns value at the specified row and column. Row and column are offset from 1.

File Output

The following statements are used to control file output.

\[
\text{open MODE FILENAME}
\]

The above statement opens the file \text{FILENAME}. \text{MODE} is one of the following:

- \text{create} — Creates the file if it doesn't exist or overwrites if it does.
- \text{append} — Appends to an existing file. Generates an error if the file does not exist.

This returns the file handle for the opened file.

\[
\text{write FILE TEXT_EXPRESSION}
\]

\[
\text{writeln FILE TEXT_EXPRESSION}
\]

The above expressions write the text expression to the file. The first statement writes the text as is while the second also writes a new line after the expression has been written. It generates an error if \text{FILE} is not an open file object.

\[
\text{close FILE}
\]

This closes the file \text{FILE}.

You could use the commands above to open a file and generate text output directly:

\[
\text{set file = open create 'C:/Data/script.out'}
\text{for I from 1 to 3}
\text{\quad write file 'Stream ' << I}
\text{endfor}
\text{close file}
\]
Exit Commands

The following commands are used for existing scripts and Clementine.

```plaintext
exit current
exit current CODE

Exit from the current script with the optional exit code (default is 0). If there are no additional scripts to execute, this command exits batch mode.

exit Clementine
exit Clementine CODE

Exit from Clementine with the optional exit code (default is 0).
```

Controlling Script Execution

Script execution normally processes one statement after another. However, you can override this execution order by using a conditional if statement and several varieties of for loops.

```plaintext
if EXPR then STATEMENTS 1
else STATEMENTS 2
endif

If EXPR is a Boolean expression that evaluates to true, then this script executes STATEMENTS 1. Otherwise, this script executes STATEMENTS 2. The else clause is optional—for example:

```plaintext
if s.maxsize > 10000 then
s.maxsize = 10000
connect s to :derive
endif
```
The for loop has a variety of forms:

```
for PARAMETER in LIST
STATEMENTS
endfor
```

This script executes STATEMENTS once for each value in LIST assigned to PARAMETER, using the order of the list. The list has no surrounding brackets, and its contents are constants.

```
for PARAMETER from N to M
STATEMENTS
endfor
```

This script executes STATEMENTS once for each integer between N and M, inclusive, assigned to PARAMETER.

```
for PARAMETER in_models
STATEMENTS
endfor
```

The above script executes STATEMENTS once for each model currently on the generated models palette with the name of the model assigned to PARAMETER.

```
for PARAMETER in_fields_at NODE
STATEMENTS
endfor
```

The above script executes STATEMENTS once for each field available on the downstream side of NODE with the field name assigned to PARAMETER.

```
for PARAMETER in_fields_to NODE
   STATEMENTS
endfor
```

The above script executes STATEMENTS once for each field on the upstream side of NODE assigned to PARAMETER.

```
exit
```

```
for PARAMETER in_streams
   STATEMENTS
```
The above script executes STATEMENTS once for each loaded stream palette assigned to PARAMETER. If PARAMETER is the special variable stream then the current stream is set for STATEMENTS in the loop. The original value of stream is restored when the loop terminates.

The above command exits from the current script. If there are no additional scripts to execute, this command exits batch mode.

exit Clementine

The above command exits the Clementine application.

**Executing and Interrupting Scripts**

In Clementine, you can execute and interrupt scripts. You can execute a script using any of the following methods:

- Click the Execute button (marked with a green arrow) within a scripting dialog box.
- Execute a stream where Run this script is set as the default execution method.
- Use the -execute flag on startup, either in normal or batch mode.

*Note:* A SuperNode script is executed when the SuperNode is executed as long as you have selected Run this script within the SuperNode script dialog box.

When you are not in batch mode, during script execution the red Stop button is activated in the Scripting dialog box toolbar. Using this button, you can abandon the execution of the script and any current stream.
**Command Line Arguments**

**Invoking the Software**

Using the command line of your operating system, you can launch either the Clementine user interface (client machine only) or Clementine in batch mode (client or server machines). To launch Clementine or Clementine Batch:

- Open a DOS window or command prompt window.
- Type the command `clemb` or `clementine` as well as any arguments (flags) used to load streams, execute scripts, and connect to a server.

**Command Line Arguments**

There are a number of command line arguments, also referred to as flags, that you can use to alter the invocation of Clementine or Clementine Batch. These flags are simply appended to the initial command `clementine` or `clemb`.

For example, you can combine the `clemb` command with other startup flags, such as `-stream` and `-execute`, in order to load and execute streams in batch mode. The following command loads and executes the stream `report.str` without invoking the user interface:

```
clemb -stream report.str -execute
```

You can also execute Clementine states and scripts in this manner, using the `-state` and `-script` flags, respectively.
Combining Multiple Arguments

Multiple arguments can be combined in a single command file specified at invocation using the @ symbol followed by the filename. This enables you to shorten the clementine or clemb invocation and overcome any operating system limitations on command length. For example, the following startup command starts Clementine Batch using all of the arguments specified in the file referenced by <commandFileName>.

clemb @<commandFileName>

A command file can contain all arguments previously specified individually at startup. For example, the command line invocation

clemb -stream report.str -Porder.full_filename=APR_orders.dat -Preport.filename=APR_report.txt -execute

may be replaced by a command file with one argument per line as follows:

- stream report.str
- Porder.full_filename=APR_orders.dat
- Preport.filename=APR_report.txt
- execute

When writing and referencing command files, be sure to follow these constraints:

- Use only one command per line.
- A command file cannot contain an embedded @CommandFile argument.
- Enclose the filename and path in quotes if spaces are required—for example, clementine @"c:/Program Files/clementine/scripts/my command file.txt".

Server Connection Arguments

In order to complete execution requests, Clementine (in both user interface and batch modes) connects to a server. If you have not specified a server using the arguments listed below, the application will connect to a default server. The -local and -server flags are used to override the default server connection from the command line.
The `-server` flag tells Clementine that it should connect to a public server, and the flags `-hostname`, `-port`, `-username`, `-password`, and `-domain` are used to tell Clementine how to connect to the public server.

The `-local` flag tells Clementine to launch its own local execution server (clemlocal). In this mode, the server isn't public and is used only by the current session of Clementine.

For more information on arguments used to run in local or server mode, consult the following table.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-local</code></td>
<td>Overrides saved settings and runs Clementine in local mode, using its own local execution server (clemlocal). If <code>-server</code> is also specified, the application will fail and a warning will be raised.</td>
</tr>
<tr>
<td><code>-server</code></td>
<td>Overrides saved settings and runs Clementine in server mode, connecting to a public server using the flags <code>-hostname</code>, <code>-port</code>, <code>-username</code>, <code>-password</code>, and <code>-domain</code>. If <code>-client</code> is also specified, the application will fail and a warning will be raised.</td>
</tr>
<tr>
<td><code>-hostname &lt;name&gt;</code></td>
<td>The host name of the server machine. Available in server mode only.</td>
</tr>
<tr>
<td><code>-port &lt;number&gt;</code></td>
<td>The port number of the specified server. Available in server mode only.</td>
</tr>
<tr>
<td><code>-username &lt;name&gt;</code></td>
<td>The user name with which to log in to the server. Available in server mode only.</td>
</tr>
<tr>
<td><code>-password &lt;password&gt;</code></td>
<td>The password with which to log in to the server. Available in server mode only. <em>Note:</em> If the <code>-password</code> argument is not used, you will be prompted for a password.</td>
</tr>
<tr>
<td><code>-epassword &lt;encodedpasswordstring&gt;</code></td>
<td>The encoded password with which to log in to the server. Available in server mode only. <em>Note:</em> An encoded password can be generated from the Tools menu of the Clementine application.</td>
</tr>
<tr>
<td><code>-domain &lt;name&gt;</code></td>
<td>The domain used to log in to the server. Available in server mode only.</td>
</tr>
<tr>
<td><code>-P &lt;name&gt;=&lt;value&gt;</code></td>
<td>Used to set a startup parameter. Can also be used to set node properties, or slot parameters.</td>
</tr>
</tbody>
</table>
## System Arguments

The following table describes system arguments available for both command line invocation of the user interface and Clementine Batch.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>@&lt;commandFile&gt;</td>
<td>The ‘@’ character followed by a filename specifies a command list. When clemb or clementine encounters an argument beginning with the character ‘@’, it operates on the commands in that file as if they had been on the command line. For more information, see “Combining Multiple Arguments” on page 732.</td>
</tr>
<tr>
<td>-directory &lt;dir&gt;</td>
<td>Sets the default working directory. In local mode, this directory is used for both data and output.</td>
</tr>
<tr>
<td>-server_directory &lt;dir&gt;</td>
<td>Sets the default server directory for data. The working directory, specified using the -directory flag, is used for output.</td>
</tr>
<tr>
<td>-execute</td>
<td>After starting, execute any stream, state, or script loaded at startup. If a script is loaded in addition to a stream or state, the script alone will be executed.</td>
</tr>
<tr>
<td>-stream &lt;stream&gt;</td>
<td>At startup, load the stream specified. Multiple streams can be specified, but the last stream specified will be set as the current stream.</td>
</tr>
<tr>
<td>-script &lt;script&gt;</td>
<td>At startup, load the standalone script specified. This can be specified in addition to a stream or state as described below, but only one script can be loaded at startup.</td>
</tr>
<tr>
<td>-state &lt;state&gt;</td>
<td>At startup, load the saved state specified.</td>
</tr>
<tr>
<td>-project &lt;project&gt;</td>
<td>Load the specified project. Only one project can be loaded at startup.</td>
</tr>
<tr>
<td>-output &lt;output&gt;</td>
<td>At startup, load the saved output object (.coul format file).</td>
</tr>
<tr>
<td>-help</td>
<td>Display a list of command line arguments. When this option is specified, all other arguments are ignored and the Help screen is displayed.</td>
</tr>
<tr>
<td>-P &lt;name&gt;=&lt;value&gt;</td>
<td>Used to set a startup parameter. Can also be used to set node properties, or slot parameters.</td>
</tr>
</tbody>
</table>

*Note:* Default directories can also be set in the user interface. To access the options, from the File menu, choose Set Working Directory or Set Server Directory.
**Loading Multiple Files**

From the command line, you can easily load multiple streams, states, and outputs at startup. To do so, you should repeat the relevant argument for each object loaded. For example, to load and execute two streams called `report.str` and `train.str`, you would use the following command line arguments:

```
clemb -stream report.str -stream train.str -execute
```

**Log File Arguments**

Running Clementine in batch mode produces a log file. By default, the name of this log file is `clem_batch.log`, but you can specify an alternative name using the `-log` flag. For example, the following command executes `report.str` in batch mode and sends the logging information to `report.log`:

```
clemb -stream report.str -execute -log report.log
```

Normally, the log file overwrites any existing file of the same name, but you can make Clementine append to the log file instead by using the `-appendlog` flag. Logging can also be suppressed altogether by using the `-nolog` flag.

*Note:* Logging arguments are available only when running in batch mode.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>-appendlog</td>
<td>Append to an existing log file instead of creating a new one (batch mode only).</td>
</tr>
<tr>
<td>-log &lt;logfile&gt;</td>
<td>Specify a log file for batch mode logging information instead of the default <code>clem_batch.log</code>. <em>Note:</em> Specifying <code>stdout</code> will cause logging to the console.</td>
</tr>
<tr>
<td>-nolog</td>
<td>Produce no logging information in batch mode.</td>
</tr>
</tbody>
</table>

**Parameter Arguments**

Parameters can be used as flags during command line execution of Clementine or Clementine Batch mode. In command line arguments, the `-P` flag is used to denote a parameter of the form `-P<name>=<value>`. 
Parameters can be any of the following:

- **Simple parameters**, or parameters used directly in CLEM expressions.
- **Slot parameters**, also referred to as node properties. These parameters are used to modify the settings of nodes in the stream.
- **Command line parameters**, which are parameters used to alter the invocation of Clementine or Clementine Batch.

For example, you can supply data source user names and passwords as command line flags, such as the following:

```clementine -stream response.str -P:databasenode.username=george -P:databasenode.password=jetson```

For more information, see “Using Parameters in Batch Mode” in Chapter 17 on page 594.
Node and Stream Properties

Properties Reference Overview

Node and stream properties allow you to specify options for a variety of nodes, such as Filter, Multiplot, Neural Net, and Evaluation. Other types of properties refer to high-level stream operations, such as caching or SuperNode behavior. Using a combination of scripting commands, parameters, and node and stream properties, you can automate a number of operations and run Clementine in batch mode.

- Node properties can be used in SuperNode parameters. For more information, see “Using SuperNode Parameters to Access Node Properties” in Chapter 15 on page 569.
- Node properties can also be used as part of a command line option (using the -P flag) when starting Clementine. For more information, see “Using Parameters in Batch Mode” in Chapter 17 on page 594.

This section describes the node settings that you can control using properties and how you can reference these properties, including:

- Properties syntax
- Examples of node and stream properties
- Structured node properties

In the context of scripting within Clementine, node and stream properties are often called slot parameters. In this guide, they are referred to as node or stream properties.
Syntax for Node and Stream Properties

Properties must use the following syntax structure:

NAME:TYPE.PROPERTY

where NAME is the name of a node, and the TYPE is its type, such as multiplotnode or derivenode. You can omit either NAME or TYPE, but you must include at least one of them. PROPERTY is the name of the node or stream parameter that your expression refers to. For example, the following syntax is used to filter the Age field from downstream data:

set mynode:filternode.include.Age = false

To use a custom value for any of the parameters (NAME, TYPE, or PROPERTY), first set the value in a statement, such as set derive.newname = mynewfield. From that point on, you can use the value, mynewfieldname, as the parameter by preceding it with the ^ symbol. For example, you can set the type for the Derive node named above using the following syntax:

set ^mynewfield.result_type = "Conditional"

All nodes used in Clementine can be specified in the TYPE parameter of the syntax NAME:TYPE.PROPERTY. Additionally, any node defined by the Clementine External Module Interface can also be controlled using scripting parameters.

Structured Properties

There are two ways in which scripting uses structured properties for increased clarity when parsing:

- To give structure to the names of properties for complex nodes, such as Type, Filter, or Balance nodes.
- To provide a format for specifying multiple properties at once.
Structuring for Complex Interfaces

The scripts for nodes with tables and other complex interfaces, such as the Type, Filter, and Balance nodes, must follow a particular structure in order to parse correctly. These structured properties need a name that is more complex than the name for a single identifier. For example, within a Filter node, each available field (on its upstream side) is switched either on or off. In order to refer to this information, the Filter node stores one item of information per field (whether each field is true or false), and these multiple items are accessed and updated by a single property called field. This may have (or be given) the value true or false. Suppose that a Filter node named mynode has (on its upstream side) a field called Age. To switch this off, set the property mynode.include.Age to the value false, as follows:

```
set mynode.include.Age = false
```

Structuring to Set Multiple Properties

For many nodes, you can assign more than one node or stream property at a time. This is referred to as the multiset command or set block. For more information, see “Setting Properties and Parameters” in Appendix B on page 700.

In some cases, a structured property can be quite complex. The backslash (\) character can be used as line continuation character to help you line up the arguments for clarity. For example:

```
mynode:sortnode.keys = [{ 'K' Descending} \  
                      { 'Age' Ascending}\  
                      { 'Na' Descending }]
```

Another advantage of structured properties is the ability to set several properties on a node before the node is stable. By default, a multiset sets all properties in the block before taking any action based on an individual property setting. For example, when defining a Fixed File node, using two steps to set field properties would result in errors because the node is not consistent until both settings are valid. Defining properties as a multiset circumvents this problem by setting both properties before updating the data model.
Appendix D

Abbreviations

Standard abbreviations are used throughout the syntax for node properties. Learning the abbreviations may help you in constructing scripts.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>absolute value</td>
</tr>
<tr>
<td>len</td>
<td>length</td>
</tr>
<tr>
<td>min</td>
<td>minimum</td>
</tr>
<tr>
<td>max</td>
<td>maximum</td>
</tr>
<tr>
<td>correl</td>
<td>correlation</td>
</tr>
<tr>
<td>covar</td>
<td>covariance</td>
</tr>
<tr>
<td>num</td>
<td>number or numeric</td>
</tr>
<tr>
<td>pct</td>
<td>percent or percentage</td>
</tr>
<tr>
<td>transp</td>
<td>transparency</td>
</tr>
<tr>
<td>xval</td>
<td>cross-validation</td>
</tr>
<tr>
<td>var</td>
<td>variance or variable (in Source nodes)</td>
</tr>
</tbody>
</table>

Node and Stream Property Examples

Node and stream properties can be used in a variety of ways with Clementine. They are most commonly used as part of a script, either a **standalone script**, used to automate multiple streams or operations, or a **stream script**, used to automate processes within a single stream. SuperNode parameters can also be specified using the properties for nodes within the SuperNode. At the most basic level, properties can also be used as a command line option for starting Clementine. Using the `-p` argument as part of command line invocation, you can change a setting in the stream using a stream property.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.max_size</td>
<td>Refers to the property <code>max_size</code> of the node named <code>s</code>.</td>
</tr>
<tr>
<td>s:sampenode.max_size</td>
<td>Refers to the property <code>max_size</code> of the node named <code>s</code>, which must be a sample node.</td>
</tr>
<tr>
<td>:sampenode.max_size</td>
<td>Refers to the property <code>max_size</code> of the Sample node in the current stream (there must be only one Sample node).</td>
</tr>
</tbody>
</table>
Node and Stream Properties

<table>
<thead>
<tr>
<th>s:sample.max_size</th>
<th>Refers to the property max_size of the node named s, which must be a Sample node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>t.direction.Age</td>
<td>Refers to the direction of the field Age in the Type node t.</td>
</tr>
<tr>
<td>.:max_size</td>
<td>*** NOT LEGAL *** You must specify either the node name or the node type.</td>
</tr>
</tbody>
</table>

The example s:sample.max_size illustrates that you do not need to spell out node types in full.

The example t.direction.Age illustrates that some slot names can themselves be structured—in cases where the attributes of a node are more complex than simply individual slots with individual values. Such slots are called **structured** or **complex** properties.

**Node Properties**

Each type of node has its own set of legal properties. Each property has a type; this may be a general type—number, flag, or string—in which case, settings for the property are coerced to the correct type, or an error is raised if they cannot be coerced. Alternatively, the property reference may specify the range of legal values such as Discard, PairAndDiscard, and IncludeAsText, in which case an error is raised if any other value is used. Flag properties can be set to false using any of the following values: Off, OFF, off, No, NO, no, n, N, f, F, false, False, FALSE, or 0. All other values are regarded as true. In the reference tables found in this guide, the structured properties are indicated as such in the Property Description column, and their usage formats are given.

**Note:** Nodes created by the Clementine External Module Interface also have properties created for them automatically.

**Common Node Properties**

A number of properties are common to all nodes (including SuperNodes) in Clementine.
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use_custom_name</td>
<td>flag</td>
<td>This is a read-only property. You can use it in scripting to read the name (either auto or custom) for a node on the canvas.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>Use to specify a custom name for the node.</td>
</tr>
<tr>
<td>custom_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>tooltip</td>
<td>string</td>
<td>This is a read-only property. You can use this property for additional specificity when referring to a node name in scripting. For example, instead of referring to a node only by name, such as real_income, you can also specify the type, such as userinputnode or filternode.</td>
</tr>
<tr>
<td>annotation</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>cache_enabled</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>variablefilenode</td>
<td>skip_header</td>
<td>number</td>
</tr>
<tr>
<td></td>
<td>num_fields_auto</td>
<td>flag</td>
</tr>
<tr>
<td></td>
<td>num_fields</td>
<td>number</td>
</tr>
<tr>
<td></td>
<td>delimit_space</td>
<td>flag</td>
</tr>
<tr>
<td></td>
<td>delimit_tab</td>
<td>flag</td>
</tr>
<tr>
<td></td>
<td>delimit_new_line</td>
<td>flag</td>
</tr>
<tr>
<td></td>
<td>delimit_non_printing</td>
<td>flag</td>
</tr>
<tr>
<td></td>
<td>delimit_comma</td>
<td>flag</td>
</tr>
<tr>
<td></td>
<td>delimit_other</td>
<td>flag</td>
</tr>
<tr>
<td></td>
<td>other</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>decimal_symbol</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Period</td>
</tr>
<tr>
<td></td>
<td>multi_blank</td>
<td>flag</td>
</tr>
<tr>
<td></td>
<td>read_field_names</td>
<td>flag</td>
</tr>
<tr>
<td></td>
<td>strip_spaces</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Right</td>
</tr>
<tr>
<td></td>
<td>invalid_char_mode</td>
<td>Discard</td>
</tr>
<tr>
<td></td>
<td>invalid_char_replacement</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>lines_to_scan</td>
<td>number</td>
</tr>
<tr>
<td></td>
<td>quotes_1</td>
<td>Discard</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IncludeAsText</td>
</tr>
<tr>
<td></td>
<td>quotes_2</td>
<td>Discard</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IncludeAsText</td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
</tr>
<tr>
<td>fixedfilenode</td>
<td>record_len</td>
<td>number</td>
</tr>
<tr>
<td></td>
<td>line_oriented</td>
<td>flag</td>
</tr>
</tbody>
</table>
## Appendix D

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>decimal_symbol</td>
<td>Default</td>
<td>For example, set :fixed-filenode.decimal_symbol = Period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Comma</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Period</td>
<td></td>
</tr>
<tr>
<td></td>
<td>skip_header</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lines_to_scan</td>
<td>number</td>
<td>For example, set :fixed-filenode.lines_to_scan = 50.</td>
</tr>
<tr>
<td></td>
<td>fields</td>
<td>list</td>
<td>Structured property. Usage format: fixedfilenode.fields = [{field start length} {field start length}]</td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td>Full name of file to be read, including directory.</td>
</tr>
<tr>
<td></td>
<td>strip_spaces</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Left</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Right</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Both</td>
<td></td>
</tr>
<tr>
<td></td>
<td>invalid_char_mode</td>
<td>Discard</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Replace</td>
<td></td>
</tr>
<tr>
<td></td>
<td>invalid_char_replacement</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>databasenode</td>
<td>mode</td>
<td>Table Query</td>
<td></td>
</tr>
<tr>
<td></td>
<td>datasource</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>username</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>password</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>epassword</td>
<td>string</td>
<td>This slot is read-only during execution. To generate an encoded password, use the Password Tool available from the Tools menu. For more information, see “Generating an Encoded Password” on page 746.</td>
</tr>
<tr>
<td></td>
<td>tablename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>strip_spaces</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Left</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Right</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Both</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_quotes</td>
<td>AsNeeded</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Always</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Never</td>
<td></td>
</tr>
<tr>
<td></td>
<td>query</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>sasimportnode</td>
<td>format</td>
<td>Windows</td>
<td>Used to describe format of the import file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNIX</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAS7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>member_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>read_formats</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_format_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_var_labels</td>
<td>flag</td>
<td></td>
</tr>
</tbody>
</table>
Appendix D

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spssimportnode</td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>spssnode</td>
<td>use_var_labels</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_value_labels</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>userinputnode</td>
<td>values</td>
<td></td>
<td>Structured property of the form: set :userinputnode.values.Age = '10,70,5'. Note: Setting values for a field that isn't present creates that field. Additionally, setting the values for a field to an empty string (&quot; &quot;) removes the specified field.</td>
</tr>
</tbody>
</table>

Generating an Encoded Password

A tool is available through the user interface to generate encoded passwords based on the Blowfish algorithm (see [http://www.counterpane.com/blowfish.html](http://www.counterpane.com/blowfish.html) for more details). Once encoded, you can copy and store the password to script files and command line arguments. The node property epassword used for databasenode and databaseexportnode stores the encoded password.

- To generate an encoded password, from the Tools menu choose:
  Encode Password

**Figure D-1**

*Encode Password Tool*

- Specify a password in the Password textbox.
- Click Encode to generate a random encoding of your password.
Click the Copy icon to copy the encoded password to the clipboard.

Paste the password to the desired script or parameter.

Encoded passwords can be used in:
- Node properties for Database Source and Output nodes.
- Command line arguments for logging into the server.
- Database connection properties stored in a .par file, the parameter file generated by a Publisher node.

**Record Operations Nodes**

The following table describes the properties available for Record Operations nodes.
### Appendix D

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>samplenode</td>
<td>mode</td>
<td>Include Discard</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sample_type</td>
<td>First OneInN RandomPct</td>
<td></td>
</tr>
<tr>
<td></td>
<td>first_n</td>
<td>integer</td>
<td>Integer used to specify a cutoff point for contiguous data sampling.</td>
</tr>
<tr>
<td></td>
<td>one_in_n</td>
<td>number</td>
<td>Number for 1-in-N sampling.</td>
</tr>
<tr>
<td></td>
<td>rand_pct</td>
<td>number</td>
<td>Percentage random sample.</td>
</tr>
<tr>
<td></td>
<td>use_max_size</td>
<td>flag</td>
<td>Use to enable limitations on size of sample. Use maximum_size property to specify a value.</td>
</tr>
<tr>
<td></td>
<td>maximum_size</td>
<td>integer</td>
<td>Specify the largest sample to be included or discarded from the data stream. This option is redundant and therefore disabled when First or Include are selected above.</td>
</tr>
<tr>
<td></td>
<td>set_random_seed</td>
<td>flag</td>
<td>Set to true in order to use a random seed for sampling.</td>
</tr>
<tr>
<td></td>
<td>random_seed</td>
<td>integer</td>
<td>Specify a value used as random seed.</td>
</tr>
<tr>
<td>selectnode</td>
<td>mode</td>
<td>Include Discard</td>
<td></td>
</tr>
<tr>
<td></td>
<td>condition</td>
<td>string</td>
<td>Condition for including/discardng.</td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>----------</td>
<td>---------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>balancenode</td>
<td>directives</td>
<td></td>
<td>Structured to balance proportion of field values based on number specified. For example, mynode:balance.directives = [{1.2 'Age &lt; 23'} {1.3 'Drug = &quot;DrugA&quot;'}]. For more information, see the example following this table.</td>
</tr>
<tr>
<td>sortnode</td>
<td>keys</td>
<td>[{stringAscending} \ {stringDescending}]</td>
<td>If no direction is set, it is assumed to be ascending by default unless the default is specified otherwise using the property listed below. For example, mynode:sortnode.keys [{ 'K' Descending} \ { 'Age' Ascending} \ { 'Na' Descending }].</td>
</tr>
<tr>
<td></td>
<td>default_ascending</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>aggregatenode</td>
<td>keys</td>
<td>[field field ... field]</td>
<td>Maximum size of sample.</td>
</tr>
<tr>
<td></td>
<td>contiguous</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>aggregates</td>
<td></td>
<td>Structured property using the form: set :aggregatenode. aggregates.Age = [Sum Mean Min Max SDev], where the desired aggregation methods are included in the list.</td>
</tr>
<tr>
<td></td>
<td>extension</td>
<td>string</td>
<td>For example, :aggregatenode. extension = &quot;Aggregated_&quot;</td>
</tr>
<tr>
<td></td>
<td>add_as</td>
<td>Suffix</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Prefix</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inc_record_count</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>count_field</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
### Balance Node Examples

The balancing directive for Balance node properties is used to balance a proportion of field values based on the number specified. This node property uses the format `: [{number string} \ {number string} \ ... {number string}]`. For example:

```plaintext
mynode:balance.directives = [{1.2 'Age < 23'} {1.3 'Drug = "DrugA"'} ]
```

*Note: If strings (using double quotes) are embedded in the expression, they need to be preceded by the escape character "\". The "\" character is also the line continuation character, allowing you to line up the arguments for clarity.*
### Field Operations Nodes

The following table describes the properties available for field operations nodes.

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filternode</td>
<td>include</td>
<td>flag</td>
<td>Structured property for field inclusion and removal. Usage format: NODE.include. FIELDNAME. For example, set mynode: filternode.include.Age = &quot;false&quot;</td>
</tr>
<tr>
<td></td>
<td>new_name</td>
<td>string</td>
<td>For example, set mynode: filternode.new_name. Age = &quot;age&quot;</td>
</tr>
<tr>
<td>fillernode</td>
<td>fields</td>
<td>[field field field]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>replace_mode</td>
<td>Always Conditional Blank Null BlankAndNull</td>
<td></td>
</tr>
<tr>
<td></td>
<td>condition</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>replace_with</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>reclassifynode</td>
<td>mode</td>
<td>Single Multiple</td>
<td></td>
</tr>
<tr>
<td></td>
<td>replace_field</td>
<td>flag</td>
<td>Used only in Single mode.</td>
</tr>
<tr>
<td></td>
<td>field</td>
<td>string</td>
<td>Used only in Single mode.</td>
</tr>
<tr>
<td></td>
<td>new_name</td>
<td>string</td>
<td>Used only in Single mode.</td>
</tr>
<tr>
<td></td>
<td>fields</td>
<td>[field1 field2 ... fieldn]</td>
<td>Used only in Multiple mode.</td>
</tr>
<tr>
<td></td>
<td>name_extension</td>
<td>string</td>
<td>Used only in Multiple mode.</td>
</tr>
<tr>
<td></td>
<td>add_as</td>
<td>Suffix Prefix</td>
<td>Used only in Multiple mode.</td>
</tr>
</tbody>
</table>
### Appendix D

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reclassify</td>
<td></td>
<td></td>
<td>Structured property for field values. Usage format: NODE.reclassify. OLD_VALUE</td>
</tr>
<tr>
<td>use_default</td>
<td>flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pick_list</td>
<td>[string string ... string]</td>
<td></td>
<td>Allows a user to import a list of known new values to populate the drop-down list in the table. For example, set:reclassify.pick_list = [fruit dairy cereals]</td>
</tr>
<tr>
<td>binningnode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fields</td>
<td>[field1 field2 ... fieldn]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>method</td>
<td>FixedWidth EqualCount Rank SDev</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fixed_width_name_extension</td>
<td>string</td>
<td></td>
<td>Default extension is _BIN</td>
</tr>
<tr>
<td>fixed_width_add_as</td>
<td>Suffix Prefix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fixed_bin_method</td>
<td>Width Count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fixed_bin_count</td>
<td>integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fixed_bin_width</td>
<td>real</td>
<td></td>
<td></td>
</tr>
<tr>
<td>equal_count_name_extension</td>
<td>string</td>
<td></td>
<td>Default extension is _TILE</td>
</tr>
<tr>
<td>equal_count_add_as</td>
<td>Suffix Prefix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tile4</td>
<td>flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tile5</td>
<td>flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tile10</td>
<td>flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tile20</td>
<td>flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tile100</td>
<td>flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------</td>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>use_custom_tile</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>custom_tile_name_extension</td>
<td>string</td>
<td>Default extension is _TILEN</td>
</tr>
<tr>
<td></td>
<td>custom_tile_add_as</td>
<td>Suffix</td>
<td></td>
</tr>
<tr>
<td></td>
<td>custom_tile</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tied_values_method</td>
<td>Next</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rank_order</td>
<td>Ascending</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rank_add_as</td>
<td>Suffix</td>
<td>This option applies to rank, fractional rank, and percentage rank.</td>
</tr>
<tr>
<td></td>
<td>rank</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rank_name_extension</td>
<td>string</td>
<td>Default extension is _RANK</td>
</tr>
<tr>
<td></td>
<td>rank_fractional</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rank_fractional_name_extension</td>
<td>string</td>
<td>Default extension is _F_RANK</td>
</tr>
<tr>
<td></td>
<td>rank_pct</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rank_pct_name_extension</td>
<td>string</td>
<td>Default extension is _P_RANK</td>
</tr>
<tr>
<td></td>
<td>sdev_name_extension</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sdev_add_as</td>
<td>Suffix</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sdev_count</td>
<td>One</td>
<td></td>
</tr>
<tr>
<td></td>
<td>historynode</td>
<td>fields</td>
<td>[field field field]</td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>span</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unavailable</td>
<td>Discard</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fill_with</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
<td>-----------------------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>settoflagnode</td>
<td>fields_from</td>
<td>[field field field]</td>
<td>For example, set :settoflagnode.fields_from.Drug = [drugA drugB] creates flag fields called Drug_drugA and Drug_drugB.</td>
</tr>
<tr>
<td></td>
<td>true_value</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>false_value</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_extension</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>extension</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>add_as</td>
<td>Suffix, Prefix</td>
<td></td>
</tr>
<tr>
<td></td>
<td>aggregate</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>keys</td>
<td>[field field field]</td>
<td></td>
</tr>
<tr>
<td>typenode</td>
<td>direction</td>
<td>In, Out, Both, None</td>
<td>Structured property for field directions. Usage format: NODE.direction.FIELDNAME</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td>Range, Flag, Set, Typeless, Discrete, Default</td>
<td>Type of field. Setting type to Default will clear any values parameter setting, and if value_mode has the value Specify, it will be reset to Read. If value_mode is set to Pass or Read, setting type will not affect value_mode. Usage format: NODE.type.FIELDNAME</td>
</tr>
<tr>
<td></td>
<td>storage</td>
<td>Unknown, String, Integer, Real, Time, Date, Timestamp</td>
<td>Read-only structured property for field storage type. Usage format: NODE.storage.FIELDNAME</td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>check</td>
<td>None</td>
<td>Structured property for field type and range checking. Usage format: NODE.check.FIELDNAME. For a range field, the first value is the minimum and the last value is the maximum. For sets, specify all values. For flags, the first value represents false and the last value represents true. Setting this property automatically sets the value_mode property to Specify. Usage format: NODE.values.FIELDNAME.</td>
</tr>
<tr>
<td></td>
<td>values</td>
<td>[value value]</td>
<td>For a range field, the first value is the minimum and the last value is the maximum. For sets, specify all values. For flags, the first value represents false and the last value represents true. Setting this property automatically sets the value_mode property to Specify. Usage format: NODE.values.FIELDNAME.</td>
</tr>
<tr>
<td></td>
<td>value_mode</td>
<td>Read</td>
<td>Determines how values are set. Note that you cannot set this property to Specify directly; to use specific values, set the values property. Usage format: NODE.value_mode.FIELDNAME.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify</td>
<td></td>
</tr>
<tr>
<td></td>
<td>extend_values</td>
<td>flag</td>
<td>Applies when value_mode is set to Read. Set to T to add newly read values to any existing values for the field. Set to F to discard existing values in favor of the newly read values. Usage format: NODE.extend_values.FIELDNAME.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>enable_missing</td>
<td>flag</td>
<td>When set to $T$, activates tracking of missing values for the field. Usage format: NODE.enable_missing.FIELNAME</td>
<td></td>
</tr>
<tr>
<td>missing_values</td>
<td>[value value ...]</td>
<td>Specifies data values that denote missing data. Usage format: NODE.missing_values.FIELNAME</td>
<td></td>
</tr>
<tr>
<td>null_missing</td>
<td>flag</td>
<td>When set to $T$, nulls (undefined values that are displayed as $$null$ in the software) are considered missing values. Usage format: NODE.null_missing.FIELNAME</td>
<td></td>
</tr>
<tr>
<td>whitespace_missing</td>
<td>flag</td>
<td>When set to $T$, values containing only white space (spaces, tabs, and new lines) are considered missing values. Usage format: NODE.whitespace_missing.FIELNAME</td>
<td></td>
</tr>
<tr>
<td>new_name</td>
<td>string</td>
<td>Name of new field.</td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td>Single</td>
<td>Used in multiple derive mode only to select multiple fields.</td>
<td></td>
</tr>
<tr>
<td>fields</td>
<td>field field field]</td>
<td>Used in multiple derive mode only to select multiple fields.</td>
<td></td>
</tr>
<tr>
<td>name_extension</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add_as</td>
<td>Suffix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result_type</td>
<td>Formula</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>formula_expr</td>
<td>string</td>
<td>Expression for calculating a new field value in a Derive Any node.</td>
<td></td>
</tr>
<tr>
<td>flag_expr</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flag_true</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flag_false</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>set_default</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>set_value_cond</td>
<td>string</td>
<td>Structured to supply the condition associated with a given value.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Usage format: set:derivenode. set_value_cond. Retired = &quot;age &gt; 65&quot;</td>
<td></td>
</tr>
<tr>
<td>state_on_val</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>state_off_val</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>state_on_expression</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>state_off_expression</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>state_initial</td>
<td></td>
<td>On</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Off</td>
<td></td>
</tr>
<tr>
<td>count_initial_val</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>count_inc_condition</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>count_inc_expression</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>count_reset_condition</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cond_if_cond</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cond_then_expr</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cond_else_expr</td>
<td>string</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Appendix D

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reordernode</td>
<td>mode</td>
<td>Custom Auto</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sort_by</td>
<td>Name Type Storage</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ascending</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>start_fields</td>
<td>[field1 field2 … fieldn]</td>
<td>New fields are inserted after these fields.</td>
</tr>
<tr>
<td></td>
<td>end_fields</td>
<td>[field1 field2 … fieldn]</td>
<td>New fields are inserted before these fields.</td>
</tr>
</tbody>
</table>

### Graph Nodes

In addition to the node-specific options listed below, all graphs have the following appearance slots available.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>string</td>
<td>For example, “This is a title.”</td>
</tr>
<tr>
<td>title_font_name</td>
<td>string</td>
<td>For example, “Arial.”</td>
</tr>
<tr>
<td>title_font_size</td>
<td>number</td>
<td>Use to specify a font size, such as 12.</td>
</tr>
<tr>
<td>title_font_color</td>
<td>string</td>
<td>For example, “#FF00FF.”</td>
</tr>
<tr>
<td>title_bold</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>title_italic</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>caption</td>
<td>string</td>
<td>For example, “This is a caption.”</td>
</tr>
<tr>
<td>caption_font_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>caption_font_size</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>caption_font_color</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>caption_bold</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>caption_italic</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>labels_font_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>labels_font_size</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>labels_font_color</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
### Node and Stream Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>labels_bold</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>labels_italic</td>
<td>flag</td>
<td></td>
</tr>
</tbody>
</table>

**Notes**

**Turning off optional fields.** Optional fields, such as an overlay field for plots, can be turned off by setting the property value to "" (empty string). For example:

```python
set :plotnode.color_field = ""
```

**Specifying colors.** The colors for titles, captions, backgrounds, and labels can be specified using the hexadecimal strings starting with the hash (#) symbol. For example, to set the graph background to sky blue, you would use the following statement:

```python
set mygraph.graph_background="#87CEEB"
```

Here, the first two digits, 87, specify the red content; the middle two digits, CE, specify the green content; and the last two digits, EB, specify the blue content. Each digit can take a value in the range 0–9 or A–F. Together these values can specify a red-green-blue, or RGB, color. **Note:** When specifying colors in RGB, you can use the Field Chooser in the user interface to determine the correct color code. Simply hover over the color to activate a tooltip with the desired information.

When specifying colors in RGB, you can use the Field Chooser in the user interface to determine the correct color code. Simply hover over the color to activate a tooltip with the desired information.

The following table describes the specific properties available for individual Graph nodes.

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plotnode</td>
<td>x_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>three_D</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>z_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>color_field</td>
<td>field</td>
<td>Overlay field</td>
</tr>
</tbody>
</table>
### Appendix D

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>size_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>shape_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>panel_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>animation_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>transp_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_overlay_expr</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>overlay_expression</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>style</td>
<td>Point</td>
<td>Point, Line</td>
</tr>
<tr>
<td></td>
<td>point_type</td>
<td>Rectangle</td>
<td>Rectangle, Dot,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dot</td>
<td>Triangle, Hexagon,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Plus, HorizontalDash,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>VerticalDash</td>
</tr>
<tr>
<td>x_mode</td>
<td></td>
<td>Sort</td>
<td>Sort, Overlay,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AsRead</td>
</tr>
<tr>
<td>x_range_mode</td>
<td>Automatic</td>
<td>Automatic</td>
<td>Automatic, UserDefined</td>
</tr>
<tr>
<td>x_range_min</td>
<td>number</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>x_range_max</td>
<td>number</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>y_range_mode</td>
<td>Automatic</td>
<td>Automatic</td>
<td>Automatic, UserDefined</td>
</tr>
<tr>
<td>y_range_min</td>
<td>number</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>y_range_max</td>
<td>number</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>z_range_mode</td>
<td>Automatic</td>
<td>Automatic</td>
<td>Automatic, UserDefined</td>
</tr>
<tr>
<td>z_range_min</td>
<td>number</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>z_range_max</td>
<td>number</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>x_agitation</td>
<td>number</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>y_agitation</td>
<td>number</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>z_agitation</td>
<td>number</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>records_limit</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Node and Stream Properties

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>if_over_limit</td>
<td>PlotBins, PlotSample, PlotAll</td>
<td></td>
</tr>
<tr>
<td></td>
<td>title</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>caption</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x_label_auto</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x_label</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y_label_auto</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y_label</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>z_label_auto</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>z_label</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_grid</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>graph_background</td>
<td>color</td>
<td>There are standard graph colors described at the beginning of this section.</td>
</tr>
<tr>
<td></td>
<td>page_background</td>
<td>color</td>
<td>There are standard graph colors described at the beginning of this section.</td>
</tr>
<tr>
<td></td>
<td>output_to</td>
<td>Screen, BMP, JPEG, PNG, HTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

### histogramnode

<p>|                 | field                   | field     |                                           |
|                 | color_field             | field     |                                           |
|                 | panel_field             | field     |                                           |
|                 | animation_field         | field     |                                           |
|                 | range_mode              | Automatic, UserDefined |                                           |
|                 | range_min               | number    |                                           |
|                 | range_max               | number    |                                           |
|                 | bins                    | ByNumber, ByWidth |                                           |
|                 | num_bins                | number    |                                           |</p>
<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bin_width</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>normalize</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>separate_bands</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>title</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>caption</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x_label_auto</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x_label</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y_label_auto</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y_label</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_grid</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>graph_background</td>
<td>color</td>
<td>There are standard graph colors described at the beginning of this section.</td>
</tr>
<tr>
<td></td>
<td>page_background</td>
<td>color</td>
<td>There are standard graph colors described at the beginning of this section.</td>
</tr>
<tr>
<td></td>
<td>output_to</td>
<td>Screen BMP JPEG PNG HTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>distributionnode</td>
<td>x_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>color_field</td>
<td>field</td>
<td>Overlay field</td>
</tr>
<tr>
<td></td>
<td>normalize</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>plot</td>
<td>Specified Flags</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_proportional_scale</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>output_to</td>
<td>Screen Formatted Data HTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>webnode or directedwebnode</td>
<td>use_directed_web</td>
<td>flag</td>
<td>Use this parameter to create a directed web.</td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------</td>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>fields</td>
<td>[field field field]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>to_field</td>
<td>field</td>
<td></td>
<td></td>
</tr>
<tr>
<td>from_fields</td>
<td>[field field field]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>true_flags_only</td>
<td>flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>line_values</td>
<td>Absolute</td>
<td></td>
<td>OverallPct</td>
</tr>
<tr>
<td></td>
<td>OverallPct</td>
<td></td>
<td>PctLarger</td>
</tr>
<tr>
<td></td>
<td>PctLarger</td>
<td></td>
<td>PctSmaller</td>
</tr>
<tr>
<td>strong_links_heavier</td>
<td>flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>num_links</td>
<td>ShowMaximum</td>
<td></td>
<td>ShowLinksAbove</td>
</tr>
<tr>
<td></td>
<td>ShowAll</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max_num_links</td>
<td>number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>links_above</td>
<td>number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discard_links_min</td>
<td>flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>links_min_records</td>
<td>number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discard_links_max</td>
<td>flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>links_max_records</td>
<td>number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>weak_below</td>
<td>number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>strong_above</td>
<td>number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>link_size_continuous</td>
<td>flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>web_display</td>
<td>Circular</td>
<td></td>
<td>Network</td>
</tr>
<tr>
<td>title</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>caption</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>graph_background</td>
<td>color</td>
<td></td>
<td>There are standard graph colors described at the beginning of this section.</td>
</tr>
<tr>
<td>symbol_size</td>
<td>number</td>
<td></td>
<td>Used to specify a symbol size. For example, set :webnode. symbol_size = 5.5 creates</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>a symbol size larger than the default.</td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td>-----------</td>
<td>----------------------</td>
</tr>
<tr>
<td></td>
<td>output_to</td>
<td>Screen BMP JPEG PNG HTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>collectionnode</td>
<td>over_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>over_label_auto</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>over_label</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>collect_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>collect_label_auto</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>collect_label</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>three_D</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>by_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>by_label_auto</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>by_label</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>operation</td>
<td>Sum Mean Min Max SDev</td>
<td>This is a structured property with usage format: set:collectionnode.operation.Age = [Sum Mean Min Max SDev]</td>
</tr>
<tr>
<td></td>
<td>color_field</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>panel_field</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>animation_field</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>range_mode</td>
<td>Automatic UserDefined</td>
<td></td>
</tr>
<tr>
<td></td>
<td>range_min</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>range_max</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bins</td>
<td>ByNumber ByWidth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>num_bins</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bin_width</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_grid</td>
<td>flag</td>
<td></td>
</tr>
</tbody>
</table>
Node and Stream Properties

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>title</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>caption</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>graph_background</td>
<td>color</td>
<td>There are standard graph colors described at the beginning of this section.</td>
</tr>
<tr>
<td></td>
<td>page_background</td>
<td>color</td>
<td>There are standard graph colors described at the beginning of this section.</td>
</tr>
<tr>
<td></td>
<td>output_to</td>
<td></td>
<td>Screen BMP JPEG PNG HTML</td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>evalchartnode</td>
<td>chart_type</td>
<td></td>
<td>Gains Response Lift Profit ROI</td>
</tr>
<tr>
<td>evalchartnode</td>
<td>inc_baseline</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>evalchartnode</td>
<td>use_fixed_cost</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>evalchartnode</td>
<td>cost_value</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>evalchartnode</td>
<td>cost_field</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>evalchartnode</td>
<td>use_fixed_revenue</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>evalchartnode</td>
<td>revenue_value</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>evalchartnode</td>
<td>revenue_field</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>evalchartnode</td>
<td>use_fixed_weight</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>evalchartnode</td>
<td>weight_value</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>evalchartnode</td>
<td>weight_field</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>evalchartnode</td>
<td>n_tile</td>
<td></td>
<td>Quartiles Quintiles Deciles Vingtiles Percentiles 1000-tiles</td>
</tr>
<tr>
<td>evalchartnode</td>
<td>cumulative</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------</td>
<td>-----------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>style</td>
<td>Line Point</td>
<td></td>
</tr>
<tr>
<td></td>
<td>point_type</td>
<td>Rectangle Dot</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Triangle</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hexagon</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Plus</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>HorizontalDash</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VerticalDash</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_data</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>data_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>delimiter</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>new_line</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inc_field_names</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inc_best_line</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inc_business_rule</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>business_rule_expression</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_hit_condition</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hit_condition</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_score_expression</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>score_expression</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>caption_auto</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>output_to</td>
<td>Screen BMP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>JPEG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PNG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>HTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>multiplotnode</td>
<td>x_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y_fields</td>
<td>[field field field]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>panel_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>animation_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>normalize</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_overlay_expr</td>
<td>flag</td>
<td></td>
</tr>
</tbody>
</table>
### Node and Stream Properties

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>overlay_expression</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>records_limit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>if_over_limit</td>
<td>PlotBins</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PlotSample</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PlotAll</td>
<td></td>
</tr>
<tr>
<td></td>
<td>title</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>caption</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x_label_auto</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x_label</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y_label_auto</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y_label</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_grid</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>graph_background</td>
<td>color</td>
<td>There are standard graph colors described at the beginning of this section.</td>
</tr>
<tr>
<td></td>
<td>page_background</td>
<td>color</td>
<td>There are standard graph colors described at the beginning of this section.</td>
</tr>
<tr>
<td></td>
<td>output_to</td>
<td>Screen</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BMP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>JPEG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PNG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>HTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

### Modeling Nodes

The following table describes the properties available for modeling nodes.

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>neuralnetnode</td>
<td>custom_fields</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>targets</td>
<td>[field1 ... fieldN]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inputs</td>
<td>[field1 ... fieldN]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_model_name</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model_name</td>
<td>string</td>
<td>Name for new model.</td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------</td>
<td>-----------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>method</td>
<td>Quick Dynamic</td>
<td>Dynamic Multiple Prune ExhaustivePrune RBFN</td>
</tr>
<tr>
<td></td>
<td>prevent_overtrain</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>train_pct</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>set_random_seed</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>random_seed</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mode</td>
<td>Simple Expert</td>
<td></td>
</tr>
<tr>
<td></td>
<td>stop_on</td>
<td>Default Accuracy</td>
<td>Stopping mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cycles Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>accuracy</td>
<td>number</td>
<td>Stopping accuracy.</td>
</tr>
<tr>
<td></td>
<td>cycles</td>
<td>number</td>
<td>Cycles to train.</td>
</tr>
<tr>
<td></td>
<td>time</td>
<td>number</td>
<td>Time to train(minutes).</td>
</tr>
<tr>
<td></td>
<td>continue</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>show_feedback</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>binary_encode</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_last_model</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sensitivity_analysis</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>gen_logfile</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>logfile_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>initial_eta</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>high_eta</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>low_eta</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eta_decay_cycles</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hid_layers</td>
<td>One Two Three</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------</td>
<td>-----------</td>
<td>----------------------</td>
</tr>
<tr>
<td></td>
<td>hl_units_one</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hl_units_two</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hl_units_three</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>persistence</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m_topologies</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m_non_pyramids</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m_persistence</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p_hid_layers</td>
<td>One, Two, Three</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p_hid_units_one</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p_hid_units_two</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p_hid_units_three</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p_hid_rate</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p_hid_pers</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p_inp_rate</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p_inp_pers</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p_overall_pers</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_persistence</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_num_clusters</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_eta_auto</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_alpha</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_eta</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>c50node</td>
<td>custom_fields</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>target</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inputs</td>
<td>[field1 ... field2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_model_name</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model_name</td>
<td>string</td>
<td>Name for new model.</td>
</tr>
<tr>
<td></td>
<td>output_type</td>
<td>DecisionTree, RuleSet</td>
<td></td>
</tr>
<tr>
<td></td>
<td>group_symbolics</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>use_boost</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>boost_num_trials</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_xval</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xval_num_folds</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mode</td>
<td>Simple</td>
<td>Expert</td>
</tr>
<tr>
<td></td>
<td>favor</td>
<td>Accuracy</td>
<td>Generality</td>
</tr>
<tr>
<td></td>
<td>expected_noise</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_child_records</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pruning_severity</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_costs</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>costs</td>
<td></td>
<td>Structured property in the form of:       [{drugA drugB 1.5}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{drugA drugC 2.1}], where the arguments in {} are actual predicted costs.</td>
</tr>
<tr>
<td></td>
<td>use_winnowing</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_global_pruning</td>
<td>flag</td>
<td>On (true) by default.</td>
</tr>
<tr>
<td>kohonennode</td>
<td>custom_fields</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inputs</td>
<td>[field1 ... fieldN]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_model_name</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model_name</td>
<td>string</td>
<td>Custom name for new model.</td>
</tr>
<tr>
<td></td>
<td>continue</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>show_feedback</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mode</td>
<td>Simple</td>
<td>Expert</td>
</tr>
<tr>
<td></td>
<td>stop_on</td>
<td>Default</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td>time</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>length</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------</td>
<td>---------------</td>
<td>----------------------</td>
</tr>
<tr>
<td></td>
<td>decay_style</td>
<td>Linear</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exponential</td>
<td></td>
</tr>
<tr>
<td></td>
<td>phase1_neighborhood</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>phase1_eta</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>phase1_cycles</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>phase2_neighborhood</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>phase2_eta</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>phase2_cycles</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>regressionnode</td>
<td>custom_fields</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inputs</td>
<td>[field1 ... field2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>target</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_model_name</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>method</td>
<td>Enter</td>
<td>Stepwise</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Backwards</td>
<td>Forwards</td>
</tr>
<tr>
<td></td>
<td>include_constant</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_weight</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>weight_field</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mode</td>
<td>Simple</td>
<td>Expert</td>
</tr>
<tr>
<td></td>
<td>complete_records</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tolerance</td>
<td>1.0E-1</td>
<td>1.0E-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0E-3</td>
<td>1.0E-4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0E-5</td>
<td>1.0E-6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0E-7</td>
<td>1.0E-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0E-9</td>
<td>1.0E-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0E-11</td>
<td>1.0E-12</td>
</tr>
</tbody>
</table>
|                 |                    |               | Use double quotes for arguments.
<table>
<thead>
<tr>
<th>Name</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stepping_method</td>
<td>Probability Fvalue</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>probability_entry</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>probability_removal</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F_value_entry</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F_value_removal</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>selection_criteria</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>confidence_interval</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>covariance_matrix</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>collinearity_diagnostics</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>durbin_watson</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model_fit</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_squared_change</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p_correlations</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>descriptives</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>logregnode</td>
<td>custom_fields</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inputs</td>
<td>[field1 ... field2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>target</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_model_name</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>include_constant</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mode</td>
<td>Simple Expert</td>
<td></td>
</tr>
<tr>
<td></td>
<td>method</td>
<td>Enter Stepwise</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Forwards</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Backwards</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BackwardsStepwise</td>
<td></td>
</tr>
</tbody>
</table>
### Node and Stream Properties

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>model_type</td>
<td>MainEffects FullFactorial Custom</td>
<td>When FullFactorial is specified as the model type, stepping methods will not be run, even if also specified. Instead, Enter will be the method used. If the model type is set to Custom but no custom fields are specified, then a main-effects model is built.</td>
</tr>
<tr>
<td></td>
<td>custom_terms</td>
<td>[{BP Sex}/BP]/Age]</td>
<td>For example, set :logisticnode. custom_terms = [{Na} {K} {Na K}]</td>
</tr>
<tr>
<td></td>
<td>scale</td>
<td>None UserDefined Pearson Deviance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>scale_value</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>all_probabilities</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tolerance</td>
<td>1.00E-5 1.00E-6 1.00E-7 1.00E-8 1.00E-9 1.00E-10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_terms</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_max_terms</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_terms</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>probability_entry</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>probability_removal</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>requirements</td>
<td>HierarchyDiscrete HierarchyAll Containment None</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_iterations</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------</td>
<td>-----------</td>
<td>----------------------</td>
</tr>
<tr>
<td></td>
<td>max_steps</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p_converge</td>
<td>1.00E-4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.00E-5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.00E-6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.00E-7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.00E-8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>l_converge</td>
<td>1.00E-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.00E-2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.00E-3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.00E-4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.00E-5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>delta</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iteration_history</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>history_steps</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>summary</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>likelihood_ratio</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>asymptotic_correlation</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>goodness_fit</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>parameters</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>confidence_interval</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>asymptotic_covariance</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>classification_table</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>cartnode</td>
<td>custom_fields</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inputs</td>
<td>[field1 ... fieldN]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>target</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_model_name</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>levels_below_root</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_percentage</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_parent_records_pc</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_child_records_pc</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_parent_records_abs</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------</td>
<td>-------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>min_child_records_abs</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mode</td>
<td>Simple</td>
<td>Expert</td>
</tr>
<tr>
<td></td>
<td>prune_tree</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_std_err_rule</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>std_err_multiplier</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_frequency</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>frequency_field</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_weight</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>weight_field</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_impurity</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>impurity_measure</td>
<td>Gini</td>
<td>Twoing</td>
</tr>
<tr>
<td></td>
<td>max_surrogates</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>priors</td>
<td>Data</td>
<td>Equal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Custom</td>
<td>Structured property using the form: set:cartnode. custom_priors = [{drugA 0.3} {drugB 0.6}]</td>
</tr>
<tr>
<td></td>
<td>custom_priors</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>adjust_priors</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_costs</td>
<td>flag</td>
<td>Structured property in the form: [{drugA drugB 1.5} {drugA drugC 2.1}], where the arguments in {} are actual predicted costs.</td>
</tr>
<tr>
<td></td>
<td>costs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>factornode</td>
<td>custom_fields</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inputs</td>
<td>[field1 ... field2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_model_name</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model_name</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
### Appendix D

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>method</td>
<td>PC</td>
<td>ULS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>mode</td>
</tr>
<tr>
<td>max_iterations</td>
<td>max_iterations</td>
<td>number</td>
<td>complete_records</td>
</tr>
<tr>
<td>matrix</td>
<td>matrix</td>
<td>Correlation</td>
<td>extract_factors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Covariance</td>
<td>min_eigenvalue</td>
</tr>
<tr>
<td>max_factor</td>
<td>max_factor</td>
<td>number</td>
<td>rotation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delta</td>
<td>delta</td>
<td>number</td>
<td>If you select DirectOblimin as your rotation data type, you can specify a value for delta. If you do not specify a value, the default value for delta is used.</td>
</tr>
<tr>
<td>kappa</td>
<td>kappa</td>
<td>number</td>
<td>If you select Promax as your rotation data type, you can specify a value for kappa. If you do not specify a value, the default value for kappa is used.</td>
</tr>
<tr>
<td>sort_values</td>
<td>sort_values</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------</td>
<td>-----------</td>
<td>----------------------</td>
</tr>
<tr>
<td></td>
<td>hide_values</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hide_below</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>twostepnode</td>
<td>custom_fields</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inputs</td>
<td>[field1 ... fieldN]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_model_name</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>standardize</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>exclude_outliers</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cluster_num_auto</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_num_clusters</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_num_clusters</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>num_clusters</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cluster_label</td>
<td>String Number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>label_prefix</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>grinode</td>
<td>custom_fields</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>consequents</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>antecedents</td>
<td>[field1 ... fieldN]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_model_name</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_supp</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_conf</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_num_rules</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_antecedents</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>true_flags</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------</td>
<td>-----------</td>
<td>----------------------</td>
</tr>
<tr>
<td>apriorinode</td>
<td>custom_fields</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>consequents</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>antecedents</td>
<td>[field1 ... fieldN]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_model_name</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_supp</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_conf</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_antecedents</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>true_flags</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>optimize</td>
<td>Speed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_transactional_data</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>contiguous</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id_field</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>content_field</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mode</td>
<td>Simple</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Expert</td>
<td></td>
</tr>
<tr>
<td></td>
<td>evaluation</td>
<td>RuleConfidence</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DifferenceToPrior</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ConfidenceRatio</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>InformationDifference</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NormalizedChiSquare</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lower_bound</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------</td>
<td>-----------</td>
<td>----------------------</td>
</tr>
<tr>
<td>kmeansnode</td>
<td>custom_fields</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inputs</td>
<td>[field1 ... fieldN]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_model_name</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>num_clusters</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>gen_distance</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>show_proximity</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cluster_label</td>
<td>String</td>
<td>Number</td>
</tr>
<tr>
<td></td>
<td>label_prefix</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mode</td>
<td>Simple</td>
<td>Expert</td>
</tr>
<tr>
<td></td>
<td>stop_on</td>
<td>Default</td>
<td>Custom</td>
</tr>
<tr>
<td></td>
<td>max_iterations</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tolerance</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>encoding_value</td>
<td>number</td>
<td></td>
</tr>
</tbody>
</table>
### Generated Model Nodes

Generated model nodes share the same common properties as other nodes in Clementine. For more information, see “Common Node Properties” on page 741.

In addition to common node properties, there are several properties specific to generated model nodes. The table below lists these properties as well as the generated node names used for scripting.

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequencenode</td>
<td>use_model_name</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>time_field</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_time_field</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>content_fields</td>
<td>[field1 ... fieldn]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>contiguous</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_supp</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_conf</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_size</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_predictions</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mode</td>
<td>Simple Expert</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_max_duration</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_duration</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_gaps</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_item_gap</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_item_gap</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_pruning</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pruning_value</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>set_mem_sequences</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mem_sequences</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>
### Node and Stream Properties

<table>
<thead>
<tr>
<th>Generated Model</th>
<th>Node name for scripting</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Net</td>
<td>applyneuralnetnode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build C5.0</td>
<td>applyc50node</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kohonen</td>
<td>applykohonennode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear Reg.</td>
<td>applyregressionnode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logistic Reg.</td>
<td>applylogregnode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C&amp;R Tree</td>
<td>applycartnode</td>
<td>sql_generate</td>
<td>Never MissingValues NoMissingValues</td>
<td>Used to set SQL generation options during ruleset execution.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calculate_conf</td>
<td>flag</td>
<td>Available when SQL generation is enabled, this property includes confidence calculations in the generated tree.</td>
</tr>
<tr>
<td>Factor/PCA</td>
<td>applyfactornode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TwoStep</td>
<td>applytwostepnode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRI</td>
<td>applygrinode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apriori</td>
<td>applyapriorinode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kmeans</td>
<td>applykmeansnode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence</td>
<td>applysequencenode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generated Rulesets</td>
<td>applyrulenode</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Output Nodes

Output node properties are slightly different from other types of node properties. Rather than referring to a particular node options, output node properties store a reference to the output object. This is useful, for example, in taking a value from a table and then setting it as a stream parameter.

The following table describes the scripting properties available for output nodes.
<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablenode</td>
<td>full_filename</td>
<td>string</td>
<td>If disk, data, or HTML output, the name of the output file.</td>
</tr>
<tr>
<td></td>
<td>output_to</td>
<td>Screen Formatted Data, HTML, Transposed</td>
<td>Specifies location and type of output.</td>
</tr>
<tr>
<td></td>
<td>highlight_expr</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lines_per_page</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>output</td>
<td>string</td>
<td>A read-only property that holds a reference to the last table built by the node.</td>
</tr>
<tr>
<td>analysisnode</td>
<td>output_to</td>
<td>Screen, TXT, HTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>by_fields</td>
<td>[field, field]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td>If disk, data, or HTML output, the name of the output file.</td>
</tr>
<tr>
<td></td>
<td>coincidence</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>performance</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>confidence</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>threshold</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>improve_accuracy</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inc_user_measure</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>user_if</td>
<td>expr</td>
<td></td>
</tr>
<tr>
<td></td>
<td>user_then</td>
<td>expr</td>
<td></td>
</tr>
<tr>
<td></td>
<td>user_else</td>
<td>expr</td>
<td></td>
</tr>
<tr>
<td></td>
<td>user_compute</td>
<td>[Mean, Sum, Min, Max, SDev]</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------</td>
<td>---------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>matrixnode</td>
<td>fields</td>
<td>Selected Flags, Numerics</td>
<td></td>
</tr>
<tr>
<td></td>
<td>row</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>column</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cell_contents</td>
<td>CrossTabs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>function_field</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>function</td>
<td>Sum, Mean, Min, Max, SDev</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sort_mode</td>
<td>Unsorted Ascending, Descending</td>
<td></td>
</tr>
<tr>
<td></td>
<td>highlight_top</td>
<td>number</td>
<td>If non-zero, then true.</td>
</tr>
<tr>
<td></td>
<td>highlight_bottom</td>
<td>number</td>
<td>If non-zero, then true.</td>
</tr>
<tr>
<td></td>
<td>display</td>
<td>[Counts Expected, RowPct, ColumnPct, TotalPct]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>include_totals</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>output_to</td>
<td>Screen, Formatted Data, HTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lines_per_page</td>
<td>number</td>
<td></td>
</tr>
</tbody>
</table>
## Appendix D

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statisticsnode</td>
<td>output_to</td>
<td>Screen, HTML, TXT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>examine</td>
<td>[field]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>correlate</td>
<td>[field]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>statistics</td>
<td>[Count, Mean, Sum, Min, Max, Range, Variance, SDev, SErr, Median, Mode]</td>
<td>Structured property where fields to be set must be referenced with the following syntax: <code>set : setglobalsnode.globals.Age = [Sum Mean Min Max SDev]</code></td>
</tr>
<tr>
<td></td>
<td>label_correlations</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>weak_label</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>medium_label</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>strong_label</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>weak_below</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>setglobalsnode</td>
<td>strong_above</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>globals</td>
<td>[Sum, Mean, Min, Max, SDev]</td>
<td>Structured property where fields to be set must be referenced with the following syntax: <code>set : setglobalsnode.globals.Age = [Sum Mean Min Max SDev]</code></td>
</tr>
<tr>
<td></td>
<td>clear_first</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>show_preview</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>reportnode</td>
<td>output_to</td>
<td>Screen HTML TXT</td>
<td>Where to send output.</td>
</tr>
<tr>
<td></td>
<td>text</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>highlights</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>title</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lines_per_page</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>spssprocedurenode</td>
<td>syntax</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>store_data</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>store_syntax</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>store_results</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>store_path</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>output_mode</td>
<td>SPSS Clementine</td>
<td></td>
</tr>
<tr>
<td>outputfilenode or</td>
<td>full_filename</td>
<td>string</td>
<td>Name of output file.</td>
</tr>
<tr>
<td>flatfilenode</td>
<td>write_mode</td>
<td>Overwrite Append</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_field_names</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_newline_after_records</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>delimit_mode</td>
<td>Comma Tab Space Other</td>
<td></td>
</tr>
<tr>
<td></td>
<td>other_delimiter</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td></td>
<td>quote_mode</td>
<td>None Single Double Other</td>
<td></td>
</tr>
<tr>
<td></td>
<td>other_quote</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>generate_import</td>
<td>flag</td>
<td></td>
</tr>
</tbody>
</table>
### Appendix D

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>excelnode</td>
<td>create_file</td>
<td>Temporary Specified</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>file_type</td>
<td>CSV TXT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>decimal_symbol</td>
<td>Default Period Comma</td>
<td></td>
</tr>
<tr>
<td>databaseexportnode</td>
<td>datasource</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>username</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>password</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>epassword</td>
<td>string</td>
<td>This slot is read-only during execution. To generate an encoded password, use the Password Tool available from the Tools menu. For more information, see “Generating an Encoded Password” on page 746.</td>
</tr>
<tr>
<td></td>
<td>table_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>write_mode</td>
<td>Create Append</td>
<td></td>
</tr>
<tr>
<td></td>
<td>default_string_size</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>drop_existing_table</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>delete_existing_rows</td>
<td>flag</td>
<td></td>
</tr>
</tbody>
</table>
## Node and Stream Properties

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>type</td>
<td>Structured property used to set the schema type. Usage format: set :database-exportnode.type.BP = 'VARCHAR(10)'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>generate_import</td>
<td>flag</td>
<td>The following properties are advanced options for database bulk loading. A true value for Use_batch turns off row-by-row commits to the database.</td>
</tr>
<tr>
<td></td>
<td>use_batch</td>
<td>flag</td>
<td>The following properties are advanced options for database bulk loading. A true value for Use_batch turns off row-by-row commits to the database.</td>
</tr>
<tr>
<td></td>
<td>batch_size</td>
<td>number</td>
<td>Specify the number of rows to submit before manual commit to the database.</td>
</tr>
<tr>
<td></td>
<td>bulk_loading</td>
<td>Off ODBC External</td>
<td>Specify the type of bulk loading. Additional options for ODBC and External are listed below.</td>
</tr>
<tr>
<td></td>
<td>odbc_binding</td>
<td>Row Column</td>
<td>Specify row-wise or column-wise binding for bulk loading via ODBC.</td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>loader_delimit_mode</td>
<td>Tab</td>
<td>For bulk-loading via an external program, specify type of delimiter. Select Other in conjunction with <strong>loader_other_delimiter</strong> property to specify delimiters, such as the comma (,).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Space</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>loader_other_delimiter</td>
<td>string</td>
<td>A true flag activates the <strong>data_file</strong> property below, where you can specify the filename and path to write to when bulk loading to the database.</td>
</tr>
<tr>
<td></td>
<td>specify_data_file</td>
<td>flag</td>
<td>A true flag activates the <strong>data_file</strong> property below, where you can specify the filename and path to write to when bulk loading to the database.</td>
</tr>
<tr>
<td></td>
<td>data_file</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>specify_loader_program</td>
<td>flag</td>
<td>A true flag activates the <strong>loader_program</strong> property below, where you can specify the name and location of an external loader script or program.</td>
</tr>
<tr>
<td></td>
<td>loader_program</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>gen_logfile</td>
<td>flag</td>
<td>A true flag activates the <strong>logfile_name</strong> property below, where you can specify the name of a file on the server to generate an error log.</td>
</tr>
<tr>
<td></td>
<td>logfile_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------</td>
<td>-----------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>check_table_size</td>
<td>flag</td>
<td></td>
<td>A true flag allows table checking to ensure that the increase in database table size corresponds to the number of rows exported from Clementine.</td>
</tr>
<tr>
<td>loader_options</td>
<td>string</td>
<td></td>
<td>Specify additional arguments, such as <code>-comment</code> and <code>-specialdir</code>, to the loader program.</td>
</tr>
<tr>
<td>qualitynode</td>
<td>mode</td>
<td>All</td>
<td>All Selected</td>
</tr>
<tr>
<td></td>
<td>fields</td>
<td>[field1 ... fieldn]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>invalid</td>
<td>[Null Empty Space Blank]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>calculate</td>
<td>[Count Breakdown]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>output_to</td>
<td>Screen HTML TXT</td>
<td>Where to send output.</td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>spssexportnode</td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>launch_application</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>application</td>
<td>SPSS AnswerTree</td>
<td></td>
</tr>
<tr>
<td></td>
<td>field_names</td>
<td>Labels Names</td>
<td></td>
</tr>
<tr>
<td></td>
<td>generate_import</td>
<td>flag</td>
<td></td>
</tr>
</tbody>
</table>
### Appendix D

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>saselexportnode</td>
<td>format</td>
<td>Windows</td>
<td>Uniform property</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNIX</td>
<td>label fields.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAS7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAS8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>field_names</td>
<td>Names</td>
<td>Headings</td>
</tr>
<tr>
<td></td>
<td>generate_import</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>publishernode</td>
<td>published_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_data</td>
<td>FlatFile</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SPSSFile</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SASFile</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_file_full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_file_delimit_mode</td>
<td>Comma</td>
<td>Tab</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Space</td>
<td>Other</td>
</tr>
<tr>
<td></td>
<td>export_file_other_delimiter</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_file_add_newline</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_file_write_mode</td>
<td>Overwrite</td>
<td>Append</td>
</tr>
<tr>
<td></td>
<td>export_file_inc_fieldnames</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_file_quote_mode</td>
<td>None</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Double</td>
<td>Other</td>
</tr>
<tr>
<td></td>
<td>export_file_other_quote</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_file_decimal_symbol</td>
<td>Default</td>
<td>Period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Comma</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_db_datasource</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_db_username</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_db_tablename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_db_default_string_size</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------</td>
<td>-----------</td>
<td>----------------------</td>
</tr>
<tr>
<td></td>
<td>export_db_write_mode</td>
<td>Create</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Append</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_db_delete_existing_rows</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_db_drop_existing_table</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_db_type</td>
<td></td>
<td>Structured property used to set the schema type. Usage format: <code>:databaseexport.types.BP = 'VARCHAR(10)'</code></td>
</tr>
<tr>
<td></td>
<td>export_spss_full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_spss_field_names</td>
<td>Names</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Labels</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_sas_full_filenames</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_sas_format</td>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNIX</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAS7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAS8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>export_sas_field_names</td>
<td>Labels</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Names</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use_batch</td>
<td>flag</td>
<td>The following properties are advanced options for database bulk loading. A true value for Use_batch turns off row-by-row commits to the database.</td>
</tr>
<tr>
<td></td>
<td>batch_size</td>
<td>number</td>
<td>Specify the number of rows to submit before manual commit to the database.</td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>bulk_loading</td>
<td></td>
<td>Off</td>
<td>Specify the type of bulk loading. Additional options for ODBC and External are listed below.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ODBC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>External</td>
<td></td>
</tr>
<tr>
<td>odbc_binding</td>
<td></td>
<td>Row</td>
<td>Specify row-wise or column-wise binding for bulk loading via ODBC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Column</td>
<td></td>
</tr>
<tr>
<td>loader_delimit_mode</td>
<td></td>
<td>Tab</td>
<td>For bulk-loading via an external program, specify type of delimiter. Select Other in conjunction with loader_other_delimiter property to specify delimiters, such as the comma (,).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Space</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other</td>
<td></td>
</tr>
<tr>
<td>loader_other_delimiter</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>specify_data_file</td>
<td></td>
<td>flag</td>
<td>A true flag activates the data_file property below, where you can specify the filename and path to write to when bulk loading to the database.</td>
</tr>
<tr>
<td>data_file</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>specify_loader_program</td>
<td></td>
<td>flag</td>
<td>A true flag activates the loader_program property below, where you can specify the name and location of an external loader script or program.</td>
</tr>
<tr>
<td>Node</td>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>loader_program</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>gen_logfile</td>
<td>flag</td>
<td>A true flag activates the logfile_name below, where you can specify the name of a file on the server to generate an error log.</td>
</tr>
<tr>
<td></td>
<td>logfile_name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>check_table_size</td>
<td>flag</td>
<td>A true flag allows table checking to ensure that the increase in database table size corresponds to the number of rows exported from Clementine.</td>
</tr>
<tr>
<td></td>
<td>loader_options</td>
<td>string</td>
<td>Specify additional arguments, such as <code>-comment</code> and <code>-specialdir</code>, to the loader program.</td>
</tr>
<tr>
<td>dataauditnode</td>
<td>custom_fields</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fields</td>
<td>[field1 … fieldN]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>overlay</td>
<td>field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>basic_stats</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>median_stats</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>set_random_seed</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>random_seed</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>auto_sample</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_size</td>
<td>number</td>
<td></td>
</tr>
</tbody>
</table>
Appendix D

<table>
<thead>
<tr>
<th>Node</th>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>output_to</td>
<td>Screen Formatted Data HTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full_filename</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>display_graphs</td>
<td>flag</td>
<td>Used to turn on or off the display of graphs in the output matrix.</td>
</tr>
</tbody>
</table>

**SuperNode Properties**

Properties specific to SuperNodes are described in the following table. Note that common node properties also apply to SuperNodes.

<table>
<thead>
<tr>
<th>SuperNode Type</th>
<th>Property Name</th>
<th>Property Type/List of Values</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_supernode</td>
<td>parameters</td>
<td>any</td>
<td>Use this property to create and access parameters specified in a SuperNode's parameter table. See details below.</td>
</tr>
<tr>
<td>process_supernode</td>
<td>parameters</td>
<td>any</td>
<td>Use this property to create and access parameters specified in a SuperNode's parameter table. See details below.</td>
</tr>
<tr>
<td>terminal_supernode</td>
<td>parameters</td>
<td>any</td>
<td>Use this property to create and access parameters specified in a SuperNode's parameter table. See details below.</td>
</tr>
<tr>
<td></td>
<td>execute_method</td>
<td>Script Normal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>script</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
**SuperNode Parameters**

Using scripts, you can access two kinds of parameters for SuperNodes:

- Parameters, or user-defined variables, such as *Minvalue*, set for the contents of a SuperNode.

- Node properties (also called slot parameters) for nodes encapsulated within the SuperNode. For example, *Sample.rand_pct* is the parameter that accesses the random percent control for an encapsulated Sample node.

Both types must be specified in the parameter table for a SuperNode before reading or updating. For more information, see “SuperNode Parameters” in Chapter 15 on page 566.

To create or set a SuperNode parameter, use the form:

```plaintext
set mySuperNode.parameters.minvalue = 30
```

or

```plaintext
set :process_supernode.parameters.minvalue = 30
```

or

```plaintext
set :process_supernode.parameters.minvalue = "<expression>"
```

You can be even more explicit, including both name and type in the script command. For example:

```plaintext
set mySuperNode:process_supernode.parameters.minvalue = 30
```

To access properties such as *rand_pct* for encapsulated nodes within the SuperNode, be sure to include the literal name of the parameter within single quotes. For example:

```plaintext
set mySuperNode.parameters.'Data_subset:samplenode.rand_pct' = 50
```

```plaintext
set :source_supernode.parameters.'Data_subset:samplenode.rand_pct' = 50
```

*Note:* When you define parameters in SuperNodes, you must refer to parameters by their short parameter names because these are guaranteed to be unique.
Stream Properties

A variety of stream properties can be controlled by scripting. To reference stream properties, you must use a special stream variable, denoted with a `^` preceding the stream.

`sset ^stream.execute_method = Script`

Stream properties are described in the following table.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Data Type</th>
<th>Property Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>execute_method</td>
<td>Normal</td>
<td><code>Script</code></td>
</tr>
<tr>
<td>date_format</td>
<td>string</td>
<td>&quot;DDMMYY&quot; &quot;MMDDYY&quot; &quot;YYMMMDD&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;YYYYMMDD&quot; &quot;DD/MM/YY&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;DD/MM/YYYY&quot; &quot;MM/DD/YY&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;MM/DD/YYYY&quot; &quot;DD-MM-YY&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;DD-MM-YYYY&quot; &quot;MM-DD-YY&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;MM-DD-YYYY&quot; &quot;DD.MM.YY&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;DD.MM.YYYY&quot; &quot;MM-DD.MM.&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;MM.MM.MM.&quot; &quot;MM.MM.MM.&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;MM.MM.MM.&quot; &quot;DD-MM.MM.&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;DD-MM.MM.&quot; &quot;MM-DD-MM.&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;DD-MM-MM.&quot; &quot;MM-DD-MM.&quot;</td>
</tr>
<tr>
<td>date_baseline</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>date_2digit_baseline</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>Property Name</td>
<td>Data Type</td>
<td>Property Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>time_format</td>
<td>&quot;HHMMSS&quot;</td>
<td>&quot;HHMM&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;MMSS&quot;</td>
<td>&quot;HH:MM:SS&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;HH:MM&quot;</td>
<td>&quot;MM:SS&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;(H)H:(M)M:(S)S&quot;</td>
<td>&quot;(H)H:(M)M&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;(M)M:(S)S&quot;</td>
<td>&quot;HH.MM.SS&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;HH.MM&quot;</td>
<td>&quot;MM.SS&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;(H)H.(M)M.(S)S&quot;</td>
<td>&quot;(H)H.(M)M&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;(M)M.(S)S&quot;</td>
<td></td>
</tr>
<tr>
<td>time_rollover</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>import_datetime_as_string</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>decimal_places</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>decimal_symbol</td>
<td>Default</td>
<td>Period</td>
</tr>
<tr>
<td></td>
<td>Period</td>
<td>Comma</td>
</tr>
<tr>
<td>angles_in_radians</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>use_max_set_size</td>
<td>flag</td>
<td></td>
</tr>
<tr>
<td>max_set_size</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>ruleset_evaluation</td>
<td>Voting</td>
<td>FirstHit</td>
</tr>
<tr>
<td>refresh_source_nodes</td>
<td>flag</td>
<td>Use to refresh source nodes automatically upon stream execution.</td>
</tr>
<tr>
<td>script</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>annotation</td>
<td>string</td>
<td>For example, set stream.annotation = &quot;something interesting&quot;</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>For example, set x = stream.name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: This property is read only—if you want to change the name of a stream, you should save it with a different name.</td>
</tr>
</tbody>
</table>
### Property Name | Data Type | Property Description
--- | --- | ---
parameters |  | Use to update stream parameters from within a standalone script. For example, 
set ^stream.parameters.height = 23

| nodes |  | See detailed information below.

The **nodes** property is used to refer to the nodes in current stream. The following stream script provides an example:

```plaintext
var listofnodes
var thenode
set listofnodes = ^stream.nodes

set ^stream.annotation = ^stream.annotation >< "\n\nThis stream is called " >< ^stream.name >< "\n" and contains the follow

for thenode in listofnodes
set ^stream.annotation = ^stream.annotation >< "\n" >< ^thenode.node_type
endfor

The above example uses the nodes property to create a list of all nodes in the stream and write that list in the stream annotations. The annotation produced looks like this:

This stream is called "druglearn" and contains the following nodes

derivenode
neuralnetnode
variablefilenode
typenode
c50node
filternode```
CEMI Specification File Reference

Specification File Overview

The Clementine External Module Interface (CEMI) provides the ability to add external programs into the Clementine data mining system as new nodes. An external program corresponds to any program that can be run from a command line on a supported operating system.

CEMI allows the definition of what the new node looks like, what type of node it is (for example, record operations or graph node), and how Clementine must interface with the external module. All of this is done via a specification file—a text file having a prescribed structure that is interpretable by CEMI. This file begins with the word SPECFILE, ends with the word ENDSPECFILE, and contains up to three specification sections in its body.

```
SPECFILE
  <node-specification>
  <core-specification>
  <model-specification>+  
ENDSPECFILE
```

The specification file is line-oriented. In order to allow long lines to continue over multiple lines, the backslash character (\) serves as a line concatenator throughout the file. Furthermore, blank lines are ignored, and lines starting with a hash character (#) correspond to comments and are ignored. Indentation of sections is optional but is recommended to improve readability.
**Notation**

The notational scheme used to describe the structure of the specification file includes the following rules:

- “|” means “or.”
- “{” and “}” enclose the members of the set of options for an “or” group. One or more of the members should be selected.
- “+” means “zero or one instance of” (optional).
- “*” means “zero or more instances of” (multiple).
- “<” and “>” surround an identifier of a non-terminal symbol. The text between these symbols will be replaced by keywords and other specifications.

Other text appearing in the definitions of specification file sections are keywords or terminal symbols. Keywords should be specified as presented. Terminal symbols should be replaced as follows:

- <string> represents a single-quoted string.
- <word> represents an unquoted alphanumeric string.
- <integer> corresponds to an integer literal.
- <real> represents a real number literal.
- <boolean> corresponds to either “true” or “false.”

**Node Specification**

This section of the specification file defines what type of node CEMI should create and how it will appear in Clementine.

```plaintext
NODE
  NAME <word>
  TITLE <string>
  TYPE {SOURCE | PROCESS | TERMINAL | MODELLING UNREFINED+ <word> }
  PALETTE {SOURCE | FIELD | RECORD | GRAPH |
```
NAME, TITLE, TYPE, and PALETTE are all required keywords and must appear in that order. Clementine cannot parse the specification file if keywords are out of the expected order. The ICON subsection is optional.

**NAME and TITLE Keywords**

CEMI nodes have two string identifiers, NAME and TITLE, supplied in the NODE specification.

- **NAME** identifies the text appearing with the node when it is placed on the drawing area. The name can contain alphanumeric characters or underscores only, and it cannot begin with a number. In addition, the name must be unique; two nodes cannot share a name.

- **TITLE** defines what the node will be called in the relevant palette. Designate the title of the node using a single-quoted text string containing any characters.

For example, the specification:

```
NAME leximine
TITLE 'LexiMine'
```

results in a node entitled *LexiMine* on a designated palette. When placed in a stream, the name *leximine* appears under the node.

**TYPE Keyword**

Node type determines how data flows when the node is executed and is identified using TYPE in the NODE specification. There are four possible types: SOURCE, PROCESS, TERMINAL, and MODELLING. The shape of the node corresponds to the type, providing an indicator of how data flows during execution.
Source nodes provide input to a stream. CEMI source nodes import types of data that the built-in Clementine source nodes cannot handle, extending the range of supported data formats. For example, a CEMI source node could be created to read in Web log data, which contains a specific number of fields but needs a set of preprocessing steps.

Process nodes receive input from one node and send output to one or more nodes. CEMI process nodes perform new data manipulation tasks. For example, a CEMI process node could be used to incorporate new data cleaning techniques.

Terminal nodes produce output. CEMI terminal nodes create reporting tools, store data in a new way, or launch software that displays results. For example, a CEMI terminal node could be used to send data from Clementine into an Excel spreadsheet.

Modeling nodes produce either refined or unrefined models for the data at hand. Refined models can be browsed and placed in streams to process data. In contrast, unrefined models can be browsed only on the Models tab in the managers window; they cannot be used directly to process data. CEMI modeling nodes will be used for incorporating new modeling techniques. Use REFINED or UNREFINED to indicate the model type and supply a name for the new node generated on the Models tab.

**PALETTE Keyword**

Palettes provide a method for organizing nodes by functionality. Use PALETTE in the NODE specification to define the palette on which to place the CEMI node (either the default CEMI icon or a custom icon specified in the ICON subsection). The selection of a palette for the node is closely related to the node type.

SOURCE corresponds to the Sources palette, which typically contains source nodes.

RECORD yields a node on the Record Ops palette. This palette usually contains process nodes.

FIELD places the node on the Field Ops palette, which commonly holds process nodes.

GRAPH corresponds to the Graphs palette, which typically contains terminal nodes because they pass no data to subsequent nodes.
MODEL results in a node on the Modeling palette. This palette offers access to modeling nodes, which themselves generate nodes corresponding to generated models.

OUTPUT yields a node on the Output palette. Nodes on this palette are often terminal nodes, designed to display data in a tabular form.

**HASH_DATA Keyword**

When Clementine writes the data for use as input to the external module, the data can be hashed; that is, it can be put into a hash table with a representational value written. This facility limits the amount of data written to the input file and also allows external algorithms to handle internationalized data for which it was not designed, such as Chinese characters. HASH_DATA in the NODE specification allows hashing to be turned on or off. By default, HASH_DATA is false.

Hashing should be turned off when the external module makes direct references to either field names or the data set elements. For example, a search module that searches for particular text in the data will not work correctly with hashed data.

**ICON Subsection**

Custom icons used for a CEMI node are specified in the ICON subsection of the NODE specification. Custom icons can also be specified for models in the MODEL specification. In both cases, the NODE subsection is optional.

In both the NODE and MODEL specifications, icons are defined using identical syntax:

```plaintext
ICON
    STANDARD  <string>
    SMALL     <string>
ENDICON
```

The `<string>` supplied in the ICON definition corresponds to the path and filename of the custom icons.
- This string must be enclosed in single quotes and use a slash character (/) as the directory separator.

- Environment variables can be used in the filename. The example below uses $CLEO to represent the Clementine installation directory.

```xml
ICON
  STANDARD 'c:/ProgramFiles/Clementine/CEMI/images/lg_cemi_icon.gif'
  SMALL  '$CLEO/CEMI/images/sm_cemi_icon.gif'
ENDICON
```

A node in Clementine contains both a standard and small icon (as specified on the Layout tab of the Stream Properties dialog box). When specifying a custom node, be sure to include custom icons for both sizes.

- The ICON subsection will not parse correctly unless both icons are supplied.

- If either icon specified cannot be found or is not a valid image file (.gif or .jpg), a blank CEMI node icon will be displayed on the palette.

- Image size is not validated during parsing. If the icon is too large, only a portion of the image will be displayed.

*Note:* It is advised that custom icons adhere to a set of guidelines designed for seamless display in Clementine. For more information, see “Creating CEMI Node Icons” in Chapter 19 on page 627.

**Core Specification**

The main structure of the specification file is contained in the CORE section. This section defines details such as input data models and return values.

```xml
CORE
  {  <parameters-subsection>
    <execute-subsection>
    <options-subsection>+  
    <controls-subsection>
    <input-fields-subsection>+  
    <output-fields-subsection>+  
    <input-data-subsection>+  
    <output-data-subsection>+  
    <mapping-file-subsection>*  
    <return-code-subsection>+  
  }
```
Subsections for parameters, execution, options, controls, and return codes should be included for all CEMI nodes. The remaining subsections depend on the node type:

- Source nodes include subsections for output fields and output data.
- Process nodes include subsections for input fields, output fields, input data, output data, and mapping files.
- Terminal nodes include subsections for input fields, input data, mapping files, and results.
- Modeling nodes include subsections for input fields, input data, mapping files, results, and model results.

**PARAMETERS Subsection**

Parameters are defined in the PARAMETERS subsection of CORE or MODEL.

```
PARAMETERS
  { <name> <parameter-definition> | FROM <name> }*
ENDPARAMETERS
```

After each parameter name, supply a definition of the parameter. The definition includes a parameter type and a default value. The parameter type gives some semantic information about what type of value is stored by the parameter. The default value appears when the editing dialog box is first opened. Any number of parameters can be defined in PARAMETERS; however, the format for the parameter definition depends on the type. Available types include: flag, text, number, field, set, pathname, tempfile, and id.

The FROM keyword should be used only in the PARAMETERS subsection of the MODEL specification. FROM identifies parameters defined in CORE that MODEL should inherit. Follow FROM with the name of the parameter to be included in MODEL.
**Flag Parameter Definition**

A flag parameter can have one of two values: *true* or *false*. Specify the default value as unquoted text after flag.

```
<name> flag { true | false }
```

For example:

```
a_switch flag true
```

creates a flag parameter named *a_switch* that is *true* by default.

**Text Parameter Definition**

A text parameter represents any quoted string. Specify the default value as single-quoted text after text.

```
<name> text <string>
```

For example:

```
Customername text 'Smith'
```

creates a text parameter named *Customername* having a default value of *Smith*.

**Number Parameter Definition**

Use a number parameter for a real or integer value. After specifying the number type, optional lower and upper limits on the parameter value can be designated. Complete the parameter definition by supplying a numeric default value (between the lower and upper bounds, if supplied).

```
<name> number [<lower-bound> <upper-bound>] <number>
```

For example:

```
age number [0 100] 25
```
yields a number parameter named *age* that can range from 0 to 100. By default, this parameter equals 25.

If the parameter needs only a lower or upper bound, use a “–” character in place of the other bound. For example:

```
age number [0 –] 25
```

indicates that *age* cannot be smaller than 0 but has no upper bound.

Number parameters are handled internally as either real numbers or integer numbers dependent on the minimum, maximum and default values for the parameter. If a number parameter has an **integer** min, max and default then the parameter will be defined as an integer parameter and the values will be stored and treated as an integer (its control on the edit dialog will only accept integers). All other values will create a real parameter.

**Field Parameter Definition**

A field parameter contains one or more field names from the data set.

```<name> field oneof { all | numeric | symbolic }
<name> field someof { all | numeric | symbolic }
```

Follow the field designation with a keyword that indicates whether the parameter will contain one field name or multiple field names:

- **oneof**. Contains one field from the input data model.
- **someof**. Contains one or more fields from the input data model.

The definition of a field parameter concludes with a keyword designating the type of fields that the parameter references:

- **all**. The name of any field in the input data set may appear in the parameter.
- **numeric**. Restricts the available names to those for numeric fields.
- **symbolic**. Restricts the available names to those for symbolic fields.

For example:

```
inputfield1 field oneof all
```
creates a parameter named \textit{inputfield1} that contains one field name for fields of any type in the input data set. In contrast:

\begin{verbatim}
inputfield2 field someof symbolic
\end{verbatim}

yields a parameter named \textit{inputfield2} that contains one or more field names for symbolic fields in the input data set.

The field names themselves are not specified when defining the parameter. The user identifies the field names using the editing dialog box for the node.

\section*{Set Parameter Definition}

A set parameter contains one or more values from a specified set.

\begin{verbatim}
=set oneof \[ <set-members>* \] <default-value>
=set someof \[ <set-members>* \] <default-value>+
\end{verbatim}

\begin{verbatim}
<set-members> == \{ <name> | <string> | <number> \}
<default-value> == \{ <name> | <string> | <number> \}
\end{verbatim}

After the set designation, provide a keyword indicating whether the parameter will contain one value or multiple values:

- oneof. Contains one value from the set.
- someof. Contains one or more values from the set. Include the CONTROL TOGGLE keyword for the someof parameter to create a multiple check box control.

Follow the oneof or someof keyword with a bracketed, space-delimited set of possible values for the parameter. This set may contain unquoted alphanumeric strings, single-quoted strings, or numbers. The definition of a set parameter concludes by specifying the default value for the parameter selected from the set of possible values. For example:

\begin{verbatim}
payment_method set oneof [check cash credit] cash
\end{verbatim}

creates a parameter named \textit{payment_method} that contains one value from the set \{check, cash, credit\}. By default, this parameter has a value of \textit{cash}. 

**Pathname Parameter Definition**

Pathname parameters refer to file names.

```
<name>  pathname  <string>
```

This parameter allows the user to specify an existing file to edit, or to specify a new file to which to write. The default value, specified after the `pathname` designation, should include the path to the file as well as the file name. Use a slash (/) as the directory separator in paths.

Since a specification file is saved with the CEMI node, it is advisable to use environment variables when pointing to pathnames. For example, you can define an environment variable `MYAPP_PATH` and use that variable in a pathname specification such as `$MYAPP_PATH/bin/myapp.exe`.

- Use `$` to refer to environment variables in CEMI specifications.
- You may also use the special variable `$CLEO` to point to the Clementine installation directory.

Pathname parameters are fully expanded to their absolute pathnames before they are used in the execution of a CEMI node. This includes expansion of environment variables. CEMI specifications which expect relative pathnames or manipulate the contents of pathname parameters in string expressions may need to be altered to conform with this behavior.

**Tempfile Parameter Definition**

A tempfile parameter specifies the name of a temporary file.

```
<name>  tempfile  <string>
```

Supply the name for the temporary file as a single-quoted string after the `tempfile` designation. Typically, names for temporary files do not include a file suffix, allowing the name to be used as a stem for both temporary input and output files specified in the `INPUT_DATA` and `OUTPUT_DATA` subsections. For example:

```
myfile  tempfile  'temp'
```
creates a parameter named *myfile* that contains the root name *temp*, which will be used for temporary files.

**Id Parameter Definition**

An id parameter identifies the root used as a stem for the generation of temporary IDs.

```
<name> id <string>
```

A parameter of this type serves as a unique reference when needed, such as when modeling.

**EXECUTE Subsection**

In the specification file, the EXECUTE subsection defines the external executable program:

```
EXECUTE
{ COMMAND <string>  |
  COMMAND_IF <command-condition> <string>  |
  COMMAND_UNLESS <command-condition> <string> }*
ENDEXECUTE

<command-condition> ==
{ SERVER  |  <parameter-condition> }

<parameter-condition> ==
{ <word>  |  \ 
   [ <word> { <word>  |  <string>  |  <number> } ] } }
```

The string supplied in the EXECUTE subsection corresponds to the path and filename of the file to be executed by the node. This string must be enclosed in single quotes and use a slash character (/) as the directory separator. For example, a node that executes the file `C:\Program Files\Clementine\myemi.exe` would have the following EXECUTE subsection:

```
EXECUTE
  COMMAND 'c:/Program Files/Clementine/myemi.exe'
ENDEXECUTE
```
Alternatively, the string could correspond to a text parameter defined in the PARAMETERS subsection. However, in this situation, the CEMI parser requires two slashes as the directory separator. For example, to specify `myceml.exe` using a parameter:

```plaintext
PARAMETERS
  myfile text 'c://Program Files//Clementine//myceml.exe'
ENDPARAMETERS
EXECUTE
  COMMAND myfile
ENDEXECUTE
```

EXECUTE can include any number of execution commands. CEMI executes the specified programs in the order specified in the specification file.

Typically, the external executable program processes command line arguments in the specification file. All of the extra information to be included on the MS-DOS command line is listed in the OPTIONS subsection of the specification.

**Conditional Execution**

COMMAND_IF and COMMAND_UNLESS permit a single CEMI node to run different executable programs. These directives include a condition and the name of the executable program to be run. If a COMMAND_IF condition is true, the node runs the specified executable program. In contrast, a COMMAND_UNLESS condition must be false for the corresponding executable program to be run.

For example, a C5.0 CEMI node may have one executable program for producing rulesets and another for producing decision trees. A flag parameter defined in the PARAMETERS subsection can be referenced in the EXECUTE subsection to determine which executable program to run.

```plaintext
PARAMETERS
  gen_ruleset flag true
ENDPARAMETERS
EXECUTE
  COMMAND_IF [gen_ruleset] 'C:/Clementine/ruleset.exe'
  COMMAND_UNLESS [gen_ruleset] 'C:/Clementine/dec_trees.exe';
ENDEXECUTE
```

In this case, the node runs `ruleset.exe` when `gen_ruleset` is true and runs `dec_trees.exe` when `gen_ruleset` is false.
If a CEMI node needs to provide access to more than two executable programs, multiple COMMAND_IF statements could access a set parameter to determine which executable program to run. For example, a neural network CEMI node may have executable programs for five training methods: Quick, Dynamic, Multiple, Prune, or RBFN. The following specification uses the parameter training to define the method. The value of training determines which executable program to run.

PARAMETERS
  training set oneof [quick dynam mult prune rbfn] quick
ENDPARAMETERS

EXECUTE
  COMMAND_IF [training quick] 'C:/Clementine/t_quick.exe'
  COMMAND_IF [training dynam] 'C:/Clementine/t_dynam.exe'
  COMMAND_IF [training mult] 'C:/Clementine/t_mult.exe'
  COMMAND_IF [training prune] 'C:/Clementine/t_prune.exe'
  COMMAND_IF [training rbfn] 'C:/Clementine/t_rbfn.exe'
ENDEXECUTE

The executable program to run may depend on whether Clementine is running in client or server mode. When running in server mode, the executable program must be called from the server. Although it is possible to load two slightly different CEMI specification files with differently named nodes and paths of execution, this is time consuming and it produces two nodes that are practically identical.

The keyword SERVER tests whether Clementine is running in server mode. This condition returns true, allowing you to specify different executable program commands for client and server execution using COMMAND_IF and COMMAND_UNLESS. For example:

EXECUTE
  COMMAND_IF SERVER 'C:/Clementine/externs/cemi_server.exe'
  COMMAND_UNLESS SERVER 'C:/Clementine/externs/cemi_client.exe'
ENDEXECUTE

**OPTIONS Subsection**

Executable programs usually take in a list of values as input that modifies their behavior. This list typically follows the name of the executable program when invoking it from the command line. CEMI provides this functionality by passing parameter values as command line arguments.
The OPTIONS subsection is used to specify the command line arguments to be passed to the external module. The order in which the options are listed in the specification file is the order in which they will be passed to the external module. CEMI inserts spaces between the options when constructing the command line.

```
OPTIONS
  { <parameter-name> | NOPARAM } [ <string-expression> ]*
ENDOPTIONS

<string-expression> ==
  { <string> | <word> | FIELD.NAME } \n
{ >> { <string-expression> }* }
```

Begin each option definition with either an existing parameter name or the NOPARAM keyword. A parameter name corresponds to a conditional option; if the parameter is a flag that is true or if the parameter is of any other type, the corresponding option gets passed to the executable program. In contrast, NOPARAM designates unconditional options. These options are always passed to the executable program.

The option itself is a string expression enclosed in brackets that contains any text or parameter names, provided they are joined by the “><” operator. CEMI inserts the current value for the parameter in the command line. For set someof and field someof parameters, the selected items appear separated by spaces and enclosed in brackets. The values for parameters are usually set by the user in the node's editing dialog box. For example, consider the following subsections of CORE:

```
PARAMETERS
  my_text text 'ALL'
  my_set field someof all
  my_flag flag true
  cond_option set oneof [low med high] low
ENDPARAMETERS
EXECUTE
  COMMAND myce mi.exe
ENDEXECUTE
OPTIONS
  NOPARAM ['reset=' >< my_text]
  NOPARAM [my_set]
  my_flag [cond_option]
ENDOPTIONS
```

The executable program `myce mi.exe` receives at most three command line arguments:

- The value of `my_text` appended to the text “reset=”, reset=ALL.
The *my_set* field names selected from the editing dialog box, \([field1\ldots field2\ldots]\).
- The value of *cond_option*, if *my_flag* is *true*. If *my_flag* is *false*, this option is not passed.

**CONTROLS Subsection**

Controls specify how the editing dialog box associated with the node displays a particular parameter. Each parameter type has a standard way of being displayed, although it is possible to change this for some of the parameter types.

Slots displayed in the editing dialog box are specified as parameters in the CONTROLS subsection:

```plaintext
CONTROLS
  {  <basic-control> |  
      <sheet-subsection> |  
      <tab-subsection>  }*
ENDCONTROLS
```

**Basic Controls**

The definition of a control breaks down into three basic pieces:

```plaintext
{  <param> LABEL <string>+ <control-definition> |  
    NOPARAM <string>  }

<control-definition> ==
  {  CONTROL { LIST | MENU | TOGGLE | SLIDER | READFILE } +  
    VALUES [ <value-label-pair>* ] +  
    BREAK <boolean> +  
    ENABLED <parameter-condition> +  }

<value-label-pair> ==
  [  {  <word> | <string> | <number>  } <string>  }  ]
```

- **<param>**. The name of the parameter associated with the control.
- **LABEL <string>**. A single-quoted text label used to label the control in the editing dialog box. If LABEL is not included, the parameter name serves as the label.
- **<control-definition>**. Any necessary extra detail for specific parameter types, as well as details about layout and enablement.
Graphical elements are mapped to previously defined parameters; different parameters have different visual representations in the editing dialog box.

- A pathname parameter is represented by a text box for the directory and filename. It also includes a button that presents a file selection dialog box. Use CONTROL READFILE to make the file selection dialog box read-only. In this case, the external module is prevented from modifying or creating the selected file. The read-only dialog box does not allow selection of a new nonexistent file and does not allow overwriting of an existing file.

- A flag parameter is represented by a check box. If the check box is switched “on,” the corresponding parameter receives a value of true; otherwise false will be returned.

- A text parameter is represented by a text box.

- A number parameter is represented by a text box with arrows or a slider to adjust the value. To use the slider control, specify CONTROL SLIDER in the control definition.

- A field parameter is represented by a list box or menu.

- A set oneof parameter can be displayed in a list box (CONTROL LIST), a menu (CONTROL MENU), or a set of toggle buttons (CONTROL TOGGLE). For set someof parameters, the control is limited to a vertical set of check boxes; CONTROL TOGGLE changes the orientation to horizontal. The control definition also includes a list of value-label pairs in brackets. For each value in the set, specify a single-quoted label to be displayed in the control. Enclose the individual pairs and the entire list in brackets.

**Control Layout**

By default, each control appears on a new line in the editing dialog box in the order corresponding to its definitions in the specification file. To display two controls on the same line, specify BREAK false in the definition of the second parameter.

*Note:* The ALIGNED keyword, available in previous releases, is no longer supported. Controls will be aligned on the edit dialog automatically, based upon the control type and settings.
Appendix E

**Control Enablement**

To allow controls to be turned on or off based on the value of other controls, use the ENABLED keyword followed by a condition defining when to enable the control. The condition includes either a flag parameter or a set one of parameter; no other parameters are valid for enablement.

```xml
<parameter-condition> ==
  { <flag-param> | 
    [ <set-param> { <word> | <string> | <number> } ] } }
```

For a condition involving a flag parameter, the control becomes enabled when the flag parameter is true. For a condition involving a set parameter, the control becomes enabled only when the specified *label* for a set value is selected. For example, consider the following CONTROLS subsection:

```xml
CONTROLS
  my_flag LABEL 'On'
  my_text LABEL 'Enter text' ENABLED my_flag
  my_set LABEL 'Select Method' CONTROL TOGGLE \ 
    VALUES [[qu 'Quick'][dy 'Dynamic'][mu 'Multiple']] 
  my_number LABEL 'Enter time limit' ENABLED [my_set 'Quick']
ENDCONTROLS
```

The text box control for *my_text* is enabled only when the check box for *my_flag* is selected. The text box for *my_number* is enabled only when the *qu* option for *my_set* is selected, which corresponds to the label *Quick*.

**Sheets**

A sheet is an area of an editing dialog box that can be made visible or invisible based on the value of a control. By default, all controls are located on one sheet. More complex modules need more controls, so the interface can be simplified by grouping related controls on a sheet that can be hidden until needed. Create a sheet using a SHEET subsection of CONTROLS.

```xml
SHEET
  NAME <word>
  TITLE <string>
  VISIBLE <parameter-condition> +
  <basic-control-specification>*
ENDSHEET
```
The NAME value is a non-quoted string used to reference a particular sheet. TITLE defines a single-quoted string that appears at the top of the sheet in the editing dialog box.

VISIBLE defines when the sheet is available to the user. Follow this keyword with a condition involving a flag or set one of parameter. A flag parameter typically hides a panel of expert options that certain types of users do not need to see. In contrast, a condition involving a set parameter sorts controls into groups, providing access to a group when a value from the corresponding set is selected. Controls common to all values of the set should be placed in the main editing dialog box.

The definition of a sheet concludes with lines defining the controls to appear on the sheet. The basic control definition for a sheet is identical to the definition of a control in the main editing dialog box, including all layout and enablement options. The following CONTROLS subsection defines two controls on a sheet that is visible only when a flag parameter is true.

```
CONTROLS
    my_dep LABEL 'Dependent variable'
    my_indep LABEL 'Independent variable(s)'
    my_flag LABEL 'Expert Options'

SHEET
    NAME expert_sheet
    TITLE 'Expert Options'
    VISIBLE my_flag
    res_flag LABEL 'Residual plot'
    out_flag 'Outlier analysis'
ENDSHEET
ENDCONTROLS
```

The residual plot and outlier analysis can be selected only if the user requests expert options.

Sheets cannot be nested within another sheet. Each sheet can be made visible or invisible only by using a condition involving a control in the main editing dialog box. For example, the CONTROLS subsection above cannot be specified as:

```
CONTROLS
    my_dep LABEL 'Dependent variable'
    my_indep LABEL 'Independent variable(s)'
    my_flag LABEL 'Expert Options'
```
Visibility of the Outliers sheet depends on a control that appears on another sheet. To make this subsection valid, the more_flag control needs to be moved to the main editing dialog box.

**Tabs**

By default, all controls and sheets appear on the Settings tab in the editing dialog box for the CEMI node. Use a TAB subsection to create a new tab for grouping related controls.

```
TAB
   TITLE <string>
   { <basic-control-specification> | <sheet-subsection> }*
ENDTAB
```

TITLE identifies a single-quoted string used as a label for the tab. The remaining lines of the TAB subsection define the controls and sheets for the tab.

Any number of tabs can be defined, with their order in the specification file determining their order in the editing dialog box. Any controls or sheets not appearing in a TAB subsection appear on the Settings tab.

For example, suppose a CEMI node creates a simple two-dimensional scatterplot. The following specifications separate the controls for axis labeling from those for graph positioning using two tabs, Axes and Layout. The selection of fields for the plot appears on the Settings tab.

```
PARAMETERS
   xField field oneof numeric
   yField field oneof numeric
   xLab text ''
```
The INPUT_FIELDS subsection specifies which fields from the data will be used as input to the external module. Fields can be included based on their direction and/or type from the most recent Type node in the Clementine stream or based on the values held by a field type parameter. The number of fields passed can also be defined.

```plaintext
{ INCLUDE | DISCARD } \n<selection-criteria> <constraint-criteria+>
```

Specifying field selection involves:
- Defining whether to include or discard the selected fields
- Identifying the criteria used to select fields
- Imposing limits on the number of fields selected, if desired

The choice between INCLUDE and DISCARD depends on whether or not the selected fields should be passed to the external program. INCLUDE passes only the selected fields; DISCARD passes all but the selected fields. Use the latter approach if it is easier to specify the fields that should not be copied to the temporary file.
Selection Criteria

The selection criterion for INPUT_FIELDS specifies which fields to include or discard. The criterion can take several different forms:

```
{ ALL | DIRECTION [ IN+ OUT+ BOTH+ NONE+ ] | TYPE [ INTRANGE+ REALRANGE+ SET+ FLAG+ AUTO+ TYPELESS+ ] | PARAMETER <word> }
```

The keyword ALL selects all fields. INCLUDE ALL writes all fields to the temporary file. DISCARD ALL writes none of the fields, resulting in a blank file.

An alternative method for selecting fields uses direction of fields in models. DIRECTION limits the selection of fields to those having the indicated direction(s) listed in brackets. Multiple directions must be separated with a space and obey the following order: IN, OUT, BOTH, and NONE.

A third approach to field selection involves field type, which describes characteristics of the data in that field. For most types, there must be a fully instantiated Type node upstream of the CEMI node so that the information type can be accessed. Follow the TYPE keyword with a bracketed list of types to be selected. Any number of types can appear inside the brackets, but they must be separated by a space and must be included in the following order: INTRANGE, REALRANGE, SET, FLAG, AUTO, and TYPELESS.

The fourth technique uses a previously defined field parameter to designate the selected fields. The editing dialog box displays a field parameter as a list of current fields, allowing the user to select the fields to be passed. Follow the PARAMETER keyword with the name of the field parameter to select fields corresponding to the selected list.

The final method for selecting fields uses the keywords AND or OR to combine direction and type criteria for a field. AND selects a field if it has both the specified direction and type. OR, on the other hand, selects a field if it has either the specified direction or type, or both. For example:

```
INPUT_FIELDS
  INCLUDE DIRECTION [IN] AND TYPE [SET FLAG]
ENDINPUT_FIELDS
```

includes only fields with direction IN and type SET or FLAG.
**Constraint Criteria**

To impose limits on the number of fields passed to the external program through CEMI, use the keyword CHECK after the selection criteria in the INPUT_FIELDS subsection of the specification file.

CHECK { MIN <integer> + | MAX <integer> + } +

Use MIN and MAX to denote the minimum and maximum number of fields to pass. For example:

```
INPUT_FIELDS
    INCLUDE DIRECTION [OUT] CHECK MAX 1
ENDINPUT_FIELDS
```

writes at most one field having direction OUT to the temporary file passed to the external program. If multiple OUT fields exist, the first encountered appears in the temporary file. If no OUT fields occur, the file contains no fields. Alternatively:

```
INPUT_FIELDS
    INCLUDE TYPE [REALRANGE] CHECK MIN 1 MAX 5
ENDINPUT_FIELDS
```

writes at least one and no more than five fields of type REALRANGE to the temporary file.

**OUTPUT_FIELDS Subsection**

An external module may return extra new fields, remove fields, or return a completely new set of fields. Clementine needs to know what to expect in the new data model in order to check the consistency of the data on its return. The OUTPUT_FIELDS subsection relates information about the new data model. This subsection describes only the data model; the actual data must be returned to Clementine in a file referenced in the OUTPUT_DATA subsection.

```
OUTPUT_FIELDS
    { REPLACE | EXTEND }
    { <field-operation> | <process-foreach-loop> }*
ENDOUTPUT_FIELDS
```
In order to describe changes in the original data model, define the new data model using the EXTEND or REPLACE keywords.

- **EXTEND** adds zero or more new fields to the original data model and preserves the number of records, as well as the field and record order.

- **REPLACE** replaces the original data model. Use this keyword when EXTEND is not appropriate, such as when fields are removed from the original data model or when the data model is replaced entirely with new fields.

In either mode, the OUTPUT_FIELDS subsection should describe the fields that are created by the module. When using EXTEND, these are the new fields only; when using REPLACE, these are all of the fields output from the node.

New fields are described by their name and type; the field names specified in the OUTPUT_FIELDS subsection will be used in subsequent Clementine nodes, so if field names exist in the data, they are treated as data values.

**Field Operations**

Field operations consist of three parts: a creation expression, a name assignment, and a type specification.

```
<creation-exp> NAME [ <string-exp> | <param-name> ] \\
    TYPE [ <type-exp> ]
<creation-exp> ==
    { CREATE | { CREATE_IF | CREATE_UNLESS } \\
        <parameter-condition> } \\
<type-exp> ==
    { AUTO |
        AUTOSYMBOL |
        AUTONUMBER |
        AUTOSET |
        AUTOFLAG |
        REALRANGE <real-low> <real-high> |
        INTRANGE <integer-low> <integer-high> |
        FLAG <true-string> <false-string> |
        SET [ <string>* ] |
        FIELD.TYPE }
```

The creation expression determines whether the field is always generated or conditionally generated and takes one of three forms:

- **CREATE** yields unconditional field generation.
CREATE_IF generates a field only if a parameter condition is true.

CREATE_UNLESS generates a field only if a parameter condition is false.

For conditional creation, the parameter condition can contain either a flag parameter or a set parameter. For flag parameters, simply supply the parameter name. For set parameters, however, include the single-quoted label of the set value that should result in a true parameter condition.

After the creation expression, define the new field. NAME identifies the name for the field being created. Follow the keyword with brackets containing the desired name in one of the following forms:

- A single-quoted string, such as ['new_field'].
- A text parameter defined in the PARAMETERS subsection, such as [newField]. Defining a control for the parameter allows the user to specify the name for the new field using the editing dialog box.
- The keyword FIELD.NAME, which inserts the name of the current field. In FOREACH loops, the current field is the one matching the selection criteria.
- A string expression containing any combination of the above forms, concatenated using the “><” operator, such as ['$C- >< FIELD.NAME'].

TYPE defines the type for the new field. Follow the keyword with brackets containing the desired type, plus any type-specific details.

- INTRANGE and REALRANGE fields require the specification of the low and high values.
- FLAG fields require the values representing true and false to be specified.
- SET fields require a space-separated list of single-quoted members of the set.

Alternatively, use the keyword FIELD.TYPE to insert the type of the current field. In FOREACH loops, the current field is the one matching the selection criteria.

To illustrate different approaches to field creation, consider the following subsection:

```
OUTPUT_FIELDS
  EXTEND
  CREATE NAME [uncond] TYPE [AUTO]
  CREATE_IF [my_flag] NAME [cond1] TYPE [SET 'Low' 'Med' 'High']
  CREATE_UNLESS [my_set 'Type1'] NAME [cond2] \ TYPE [FLAG 'Pos' 'Neg']
ENDOUTPUT_FIELDS
```
CREATE yields a field named \textit{uncond} having a type of automatic. The file returned from the external program will always contain this field.

If \textit{my\_flag} is true, the returned file contains a field named \textit{cond1}, which is a set field containing values of \textit{Low}, \textit{Med}, and \textit{High}. If \textit{my\_flag} has a control in the editing dialog box, the user can define whether or not the file contains this new field.

If \textit{my\_set} has any value other than \textit{Type1}, the returned file contains a field named \textit{cond2}, a flag field in which \textit{Pos} represents a true value. If \textit{my\_set} has a control in the editing dialog box, the file contains \textit{cond2} when the user selects any value but \textit{Type1}.

If the program generates a fixed number of fields and the names and types of the new fields are independent of other fields, the specification file needs only one field operation line for each new field. If dependencies between fields exist, use a FOREACH loop for field creation.

\textbf{For Each Loops}

FOREACH loops provide a method of creating fields for situations in which:

\begin{itemize}
  \item The external program creates a fixed number of fields, but the names or types of the new fields depend on other fields.
  \item The data model is replaced by the external program, either by adding a variable number of new fields to the original fields or by returning only new fields.
\end{itemize}

A FOREACH loop has two general structures, depending on the boundaries of the loop.

\begin{verbatim}
FOREACH <loop-bounds>
  <field-operation>*
ENDFOREACH

<loop-bounds> ==
  { FIELD { INCLUDE | DISCARD } <selection-criteria> | FROM { <integer> | <word> } TO { <integer> | <word> } }
\end{verbatim}

The first approach creates new fields using selection criteria. In this case, add the FIELD keyword to the loop, followed by INCLUDE or DISCARD. The FOREACH loop loops through fields in the original data model, comparing each to the selection criterion, which is defined in the same way as the selection criterion for INPUT_FIELDS. INCLUDE results in the loop body being executed when a field
satisfies the selection criteria; DISCARD executes the loop for each original field that does not satisfy the selection criteria.

For example, the C5.0 model adds two fields to the data model, one for the predicted value and one for the confidence value. Each of these new fields needs a name that reflects the $OUT$ field for the model. Consider the following specification:

```
OUTPUT_FIELDS
  EXTEND
    FOREACH INCLUDE DIRECTION [OUT]
      CREATE NAME ['$C-' >< FIELD.NAME] TYPE [FIELD.TYPE]
      CREATE NAME ['$CC-' >< FIELD.NAME] TYPE [INTRANGE 0 1]
    ENDFOREACH
ENDOUTPUT_FIELDS
```

For the $OUT$ field, the loop body gets executed, creating two new fields. One field receives a name corresponding to “$C$–” appended to the name of the $OUT$ field. This field has the same type as the $OUT$ field and corresponds to the predicted value. The other new field has a name resulting from appending “$CC$–” to the $OUT$ field name and is an INTRANGE field.

If the data model is being replaced, all of the fields in the new data model must be described. Clementine does not know what the number of fields must be, so you must specify this using the second structure for FOREACH loops. This technique requires the specification of the initial and final values for the loop. Specify these bounds as either integers:

```
FOREACH FROM 0 TO 10
```

or as parameters:

```
PARAMETERS
  start_num number [0 10] 0
  end_num number [10 20] 10
ENDPARAMETERS
...
OUTPUT_FIELDS
  FOREACH FROM start_num TO end_num
...
ENDOUTPUT_FIELDS
```

The data model being described is completely new, so no information from existing fields can be used to specify the name or the types of the fields. For example:

```
OUTPUT_FIELDS
```
creates five new AUTO fields, with names field-1, field-2, and field-3. In this case, FIELD.TYPE inserts the current number of the field.

Fields created using a FOREACH loop of this type are limited to the AUTO, AUTOSYMBOL, AUTONUMBER, AUTOSET, and AUTOFLAG types.

**INPUT_DATA and OUTPUT_DATA Subsections**

When Clementine runs an external module, it creates a file to send data to the external module and/or reads a file to receive data from the external module. The INPUT_DATA and OUTPUT_DATA subsections describe the data files used by CEMI. INPUT_DATA contains information about the data sent from Clementine to the external program. Clementine creates a temporary file and copies the contents of the file referenced in this subsection to the temporary file. In contrast, OUTPUT_DATA contains information about the data sent from the external program to Clementine.

```plaintext
INPUT_DATA
   FILE_NAME { [ <param-name> | <string-expression> ] } 
   { SEPARATOR {<string> | 'TAB' } | 
     EOL <string> | 
     INC_FIELDS <boolean> | }*  
ENDINPUT_DATA
```

Any combination of SEPARATOR, EOL, and INC_FIELDS can be specified, but they must occur in that order. For output data files, change INPUT_DATA to OUTPUT_DATA and ENDINPUT_DATA to ENDOUTPUT_DATA.

FILE_NAME identifies the location and name of the data file. A pathname or tempfile parameter often supplies this information.

```plaintext
PARAMETERS
   inputfile tempfile 'C:\cemi files\infiles\temp_indata.txt'
ENDPARAMETERS
INPUT_DATA
   FILE_NAME [inputfile]
ENDINPUT_DATA
```
Alternatively, FILE_NAME can reference a single-quoted string, such as:

```plaintext
INPUT_DATA
  FILE_NAME ['C:\cemi files\infiles\temp_input_data.txt']
ENDINPUT_DATA
```

To specify a common location for multiple files, use a string appended to a tempfile or pathname parameter using the >> operator. For example:

```plaintext
PARAMETERS
  filestem pathname 'C:\cemi files\prm_infiles\'
ENDPARAMETERS
INPUT_DATA
  FILE_NAME [filestem >> 'in_data.txt']
ENDINPUT_DATA
OUTPUT_DATA
  FILE_NAME [filestem >> 'out_data.txt']
ENDOUTPUT_DATA
```

If the file location is not explicitly specified, Clementine allocates the files dynamically. However, the data source is typically used as a command option for the external module. Setting the file location allows you to ensure that the file you pass to the external module matches the name of the file created by Clementine.

The INPUT_DATA and OUTPUT_DATA subsections also describe the format of the data in the file. Controlling the format ensures that the external program and Clementine can process the data passed between them. Three keywords provide format information:

- **SEPARATOR** defines the character used to separate the field values. Supply the character within single quotes after the keyword. The default is a comma, which is used if no separator is specified. Use of the special keyword TAB separates fields in the file with a tab character.

- **EOL** identifies the character used to define the end of a record line. Denote the character using a single-quoted string after the keyword. The default is a new line character.

- **INC_FIELD** specifies whether or not the file creates contains field names on the first line. Follow the keyword with true to include field names or with false to omit them.
The external module is completely unrelated to Clementine but often needs data model information in order to process the data files sent to it. Mapping files, or metafiles, provide a description of the data model, such as that found in a fully instantiated Type node. Information included in the mapping file could be:

- A list of fields in the data
- A summary of field values
- A list of fields and their directions

The mapping file provides a translation from Clementine types to types recognized by the external module and is defined in the MAPPING_FILE subsection of the specification file. The information in the metafile can be structured in virtually an unlimited number of formats, ensuring the ability of the external module to read it. However, because data model information may not be needed by some external modules, this subsection is not required.

MAPPING_FILE

FILE_NAME { [ <param-name> | <string-expression> ] }
<map-type-subsection>
<mapping-format-subsection>
ENDMAPPING_FILE

MAPPING_FILE consists of three parts:

- The FILE_NAME keyword followed by the name of the mapping file
- The MAP_TYPE subsection, which specifies the actual translation between Clementine type descriptors and the descriptors needed by the external module
- The MAPPING_FORMAT subsection, which uses the translated types to list the data model information in a format the external module can process

The name of the mapping file can be a single-quoted string:

FILE_NAME ['c:\Clementine\mapfile.txt']

the name of a parameter defined in the PARAMETERS subsection:

FILE_NAME [mapName]
or a combination of strings and parameters joined using the "><" operator:

FILE_NAME [filePath >< 'mapfile.txt']

**MAP_TYPE Subsection**

The MAP_TYPE subsection specifies the mapping between Clementine types and the external module types. There is no guarantee that type formats used by Clementine will match those used by the external module, so this subsection allows representational mapping to be specified. For example, if the external module is written in C++, we could define a Clementine INTRANGE type as an int and a Clementine FLAG type as a string.

MAP_TYPE

```
{ FLAG => { <string> | { TRUEFIRST | FALSEFIRST } \ <delimiters> } |
  SET => { <string> | ELEMENTS <delimiters> } | |
  INTRANGE => { <string> | RANGE <delimiters> } |
  REALRANGE => { <string> | RANGE <delimiters> } |
  TYPELESS => <string> }*
```

ENDMAP_TYPE

<delimiters> ==

```
{ SEPARATOR <string> | EOL <string> }+
```

Note that no actual conversion of data is performed; only the textual description of the type is altered. In addition, in order for the types to be known, there must be a fully instantiated Type node in the stream just prior to the CEMI node. Thus, mapping files cannot be generated for CEMI source nodes. In Clementine, the following field types exist:

- **FLAG.** A set of two values, one associated with the “true” value and one with the “false” value.
- **SET.** A set of symbolic values, either strings or characters. Strings are a sequence of characters enclosed in single quotes; characters are usually an alphabetic letter but technically can be any character from the ASCII character set.
- **INTRANGE.** A set of integers, ranging from a low to a high number. The largest integer possible depends on the platform.
- **REALRANGE.** A set of reals, ranging from a low to a high number.
- **TYPELESS.** This specifies that no information is available about the field.
Appendix E

To map Clementine types, indicate the type to be mapped followed by the “=>” operator and either a single-quoted string or a keyword. Use of a string describes a field in terms of actual data types, such as:

```
MAP_TYPE
  SET => 'Varchar'
  FLAG => 'Varchar'
  INTRANGE => 'Long'
  REALRANGE => 'Double'
ENDMAP_TYPE
```

In this case, fields of type INTRANGE are described as *Long* in the mapping file.

Using a keyword instead of a string lists actual data values for each field. Each field type has its own keyword:

- For set fields, **ELEMENTS** lists every member of the set in alphabetical order, separated by commas.
- For flag fields, **TRUEFIRST** lists the true and false values, in that order, separated by a comma. To reverse the order, use **FALSEFIRST**.
- For intrange and realrange fields, **RANGE** lists the lowest and highest values separated by “..”.

Each line ends with a period.

To change the value separator or end of line character for a type, add the **SEP ARA TOR** or **EOL** keywords to the mapping definition, following each with the character to be used in single quotes. For example:

```
MAP_TYPE
  SET => ELEMENTS SEPARATOR ';' EOL '!' 
  INTRANGE => RANGE SEPARATOR '-' EOL '?
ENDMAP_TYPE
```

separates the values for set fields using a semicolon and ends the line with an exclamation point. The low and high values for intrange fields are separated by a dash, with their lines ending with a question mark.

Any types encountered that are described in the MAP_TYPE subsection are mapped to their Clementine types. For instance, if set fields are not defined in MAP_TYPE, they are mapped to *SET*. 
**MAPPING_FORMAT Subsection**

The MAPPING_FORMAT subsection generates the actual content of the mapping file. It allows the definition of a wide variety of formats in order to be as widely compatible as possible for parsing by the external module.

```
MAPPING_FORMAT
{ <map-exp> | <map-foreach-loop> }*
ENDMAPPING_FORMAT

<map-exp> ==
{ TEXT( <map-exp>+ ) | <map-exp-element> } { >> <map-exp> }*

<map-exp-element> ==
{ <word> | <string> | <number> | FIELD.TYPE | FIELD.NAME }

<map-foreach-loop> ==
FOREACH FIELD { INCLUDE | DISCARD } <selection-criteria>
{ <map-exp> }*
ENDFOREACH
```

Each map-expression line corresponds to a line of generated text. If field information is accessed, fields are referred to in order of occurrence. For example, if one map-expression line is specified, then only one line will appear in the mapping file and it will contain information from the first occurring field in the data set. To generate a line of text for each field, multiple map-expression lines are needed. Alternatively, a FOREACH FIELD loop could be used.

Construct map-expression lines using text, parameter values, or a combination or both. For example:

```
MAPPING_FORMAT
    TEXT('#CEMI Mapping File')
    TEXT('
    TEXT(infile)
    TEXT('Output File: ') >> outfile
ENDMAPPING_FORMAT
```

creates a mapping file containing four lines. The first two contain the text “*#CEMI Mapping File*” followed by a blank line. The third line identifies the input file by printing the value of the `infile` parameter. The final line appends the text “*Output File:*” to the name of the output file, which corresponds to the value of the parameter `outfile`. 
The FOREACH FIELD allows multiple lines to be inserted into the mapping file at one time by evaluating the selection criterion for each field. If the criterion is satisfied, then the map-expression line(s) within the loop are printed to the mapping file, followed by a carriage return. Use the keywords FIELD.NAME and FIELD.TYPE to insert the actual field name and field type into the mapping file. The inserted type corresponds to the mapping defined in the MAP_TYPE subsection; if this has not been specified, the Clementine type value for the field appears.

The selection criterion for a FOREACH FIELD loop has the same structure as the criterion for the INPUT_FIELDS subsection. INCLUDE applies the map-expression line to each field that satisfies the selection criterion, whereas DISCARD selects fields that do not satisfy the criterion. The criterion itself selects fields based on direction, type, a field parameter, or on a combination of the criteria (using ALL or OR). For example:

```plaintext
FOREACH FIELD DISCARD TYPE [ Flag ]
  TEXT(FIELD.NAME ' : ' FIELD.TYPE)
ENDFOREACH
```

adds the field name and type of any field that is not a flag field to the mapping file. Alternatively:

```plaintext
FOREACH FIELD INCLUDE PARAMETER usefields
  TEXT(FIELD.NAME ' : ' FIELD.TYPE)
ENDFOREACH
```

adds the name and type of all fields selected for the usefields parameter. Finally:

```plaintext
FOREACH FIELD INCLUDE PARAMETER usefields OR DIRECTION [OUT]
  TEXT(FIELD.NAME ' : ' FIELD.TYPE)
ENDFOREACH
```

adds the name and type of all selected fields for the usefields parameter, in addition to all fields that have a direction of OUT. Changing the OR to AND would limit the mapping file to all selected OUT fields for usefields.

**RETURN_CODE Subsection**

The RETURN_CODE subsection associates integers returned by the external program with text messages reflecting the state of the execution.

RETURN_CODE
SUCCESS_VALUE <integer>
 { <integer> <string> }*
ENDRETURN_CODE

This subsection of the specification file is optional. No expectations are made on the external executable program to return numeric values, but the majority of executable programs do. If the RETURN_CODE subsection is included, however, it must specify a success value using the keyword SUCCESS_VALUE followed by an integer. When the program executes successfully, it returns this value. Any integer can be used to indicate successful completion, but typically a program returns a 0 to indicate success.

Each numeric value returned by an executable program has a meaning. An explanation of what the return code means is more useful to the Clementine user than a number. Therefore, the concept of a code table is used. Each numeric code has an associated textual description, which is what will be displayed in Clementine on completion of the executable program. Define the text assigned with each possible return value by specifying the returned integer followed by a single-quoted string to display. For example:

RETURN_CODE
 SUCCESS_VALUE 0
  1 'No value has been entered for Start Field'
  2 'No value has been entered for End Field'
ENDRETURN_CODE

**RESULTS Subsection**

Terminal and modeling nodes do not return data back to the stream for further processing, so the OUTPUT_FIELDS and OUTPUT_DATA subsections of the specification file are not needed. Instead, the RESULTS subsection determines how to handle the results from the external program. This subsection defines the location and representation of the CEMI node results. For example, it specifies whether the results are a graphical display, such as a bar chart, or whether they are returned as a file, such as a text report.

RESULTS
 RESULT_TYPE { FILE_NAME [ <string-expression> ] | STD_OUT | EXTERN } |
 { RESULT_FORMAT <result-format> | RESULT_FORMAT
  <conditional-result> }+
RESULT_TYPE identifies the destination of the results from the external program. Select one of three destinations: results are written to a file, results are written to standard output, or the external module has its own display mechanism. The type of the result can take one of three forms:

- **EXTERN** indicates that the external executable program will handle the display of the results. As far as Clementine is concerned, once the external executable program has been called, it can forget about it. This type allows Clementine data to be viewed from external programs, such as spreadsheets, statistical packages, and visualization software. *Note:* Text file output cannot contain the delimiter character to output correctly.

- **FILE_NAME** defines the location of a file containing the results created by the external program. Follow the keyword with the filename and path for the results file in brackets as either a single-quoted string, a pathname or tempfile parameter, or a combination of both using the “><” operator.

- **STD_OUT** indicates that the results are sent to the standard output stream and that Clementine will display it in one of its browsers.

RESULT_FORMAT must be defined if the results are sent to a file or to the standard output stream. This specification determines how Clementine displays the results.

- **TEXT** displays the results in a text browser. If the standard output stream contains the results and they conform to the comma-delimited standard, they appear in a Clementine table.

- **HTML** shows the results in an HTML browser.
- TREE corresponds to the type of output produced by C5.0 (in DecisionTree mode) and by C&R Tree. Model output must adhere to PMML standards to be read properly in Clementine.

- RULESET corresponds to the type of output produced by C5.0 in Ruleset mode and by the Generate Ruleset option of an Association Rules node. Model output must adhere to PMML standards to be read properly in Clementine.

- ASSOC corresponds to the type of output produced by the association algorithms (such as Apriori). Model output must adhere to PMML standards to be read properly in Clementine.

- EXTERN specifies an external browser executable program. Follow the keyword with a single-quoted string expression containing the full path of the executable program.

When a numeric field is encountered in a tree or ruleset, it is split by using {>= and <} or {> and <=}. The options GREATER_THAN and LESS_THAN define which sign (>) or (<) receives the equals sign for the splits.

External programs that provide results in PMML format for TREE, RULESET, or ASSOC models have full compatibility with the Clementine model browser, including the ability to save models and generate new nodes automatically. However, the generated model represented by the results may be executed by Clementine internally. In some cases, this may not be desirable. You can override internal execution by adding an EXECUTE subsection to the MODEL specification of the specification file. The model can still be browsed in Clementine's browser.

Note: Results in the TREE, RULESET, or ASSOC formats must adhere to the PMML 2.1 standard. Earlier formats are no longer supported.

**MODEL_FILES Subsection**

External modeling modules often save generated model information in a file. CEMI can store the contents of this generated model file with the generated model node itself. This eliminates the need to maintain these external files separately because all of the necessary information to use the generated model is contained in the generated model node.

```plaintext
MODEL_FILES
    FILE_NAME { <word> | [ <string-expression> ] }*
ENDMODEL_FILES
```
Appendix E

Specify each model file using the FILE_NAME keyword, followed by the name and path of the file as either a single-quoted string, a parameter defined in the PARAMETERS subsection, or a combination of both using the “>” operator.

When the model-building node is executed, the files specified in the MODEL_FILES block are read and stored in the generated model. When the generated model is executed, the files are written back out prior to execution of the generated model executable program. Any parameters used in the MODEL_FILES subsection must be inherited by the generated model using the FROM keyword in the PARAMETERS subsection of the MODEL specification.

Model Specification

The MODEL specification describes the behavior of generated models created by the external executable program through a CEMI node of type MODELLING. In essence, a generated model is a process node, and the MODEL specification defines how this generated model communicates with Clementine.

MODEL contains the same subsections as the CORE specification but omits the CONTROLS subsection because generated models do not have an editing dialog box. Furthermore, most of the MODEL subsections are optional, whereas most CORE subsections are required. The content of the specification file for MODELLING nodes depends on whether CEMI creates an unrefined or refined model.

When executed as part of a stream, refined and unrefined modeling nodes produce models on the Models tab in the managers window. Both model types can be browsed, but only the refined generated model can be placed on the drawing area. The unrefined generated model cannot be executed and cannot be moved from the Models tab. This is because the model is not really a model, but a textual representation of the results. In contrast, the refined generated model also shows a textual representation of the results when browsed, but the format of the results allows them to generate extra modeling information when added to a stream.

In a modeling node specification file, both unrefined and refined models contain a MODEL specification. However, the unrefined model includes the MODEL specification in name only; it does not contain any information.
Unrefined Models Specification

An unrefined model cannot be executed, so very little needs to be added to the specification file for these models. The MODEL specification has the following structure:

```
MODEL
  PARAMETERS
  ENDPARAMETERS
ENDMODEL
```

The PARAMETERS subsection is empty because any specified parameters would be redundant.

The results for unrefined models are similar to the results for terminal nodes. Usually, an unrefined model contains machine-readable text. This can be browsed as TEXT or HTML, using either Clementine’s internal browsers or by specifying an external browser. Define the method for handling the results in the RESULTS subsection of CORE.

Refined Models Specification

The MODEL specification deals with the execution of the generated model, which occurs when it has been placed in a stream on the drawing area. As the data flows through the generated model, execution of the external module happens for a second time unless the EXECUTE subsection of MODEL specifies a different executable program.

For a refined model, MODEL may contain the following subsections, which have the same structure as the identically named subsections of CORE:

- PARAMETERS. Contains any new parameters needed for execution of the program. Parameters from the CORE section can be referenced here by using the FROM keyword; they will contain the same values. Any tempfile parameters, however, will have lost their values after the CORE execution because those parameters are temporary. Otherwise, the input file will retain the “train” data instead of replacing it with the “test” data.

- EXECUTE. Specifies the executable program that executes the generated model. This may be the same executable program defined in CORE.
- **OPTIONS.** Identifies command line arguments to pass to the external program defined in the EXECUTE subsection.
- **INPUT_FIELDS.** Specifies the fields sent to the generated model. These fields “test” the model.
- **INPUT DATA.** Specifies the file containing the input data.
- **OUTPUT_FIELDS.** Defines the fields returned by the generated model.
- **OUTPUT DATA.** Specifies the comma-delimited file containing the output data read into Clementine.
- **MAPPING_FILE.** Generates a file containing type node information.
- **RETURN_CODE.** Specifies error messages for the execution of the generated model.
- **MODEL_FILES.** Lists files generated during CORE execution that are needed by the generated model during MODEL execution.

### Externally Executed Generated Models

An **externally executable generated model** produces a file containing data, usually in the default comma-delimited format, when executed. Clementine reads this file back into the node that follows the refined model node in the stream.

CEMI nodes that produce these models require the **PARAMETERS**, **EXECUTE**, **OPTIONS**, **INPUT_FIELDS**, **INPUT_DATA**, **OUTPUT_FIELDS**, and **OUTPUT_DATA** subsections in the MODEL specification.

### Internally Executed Generated Models

An **internally executable generated model** uses Clementine’s internal mechanism for execution when placed in a stream. These models create rules and rule sets in a format that Clementine can use to execute them internally. Executing the generated model produces a file containing data in one of three specific result formats (**TREE**, **RULESET**, or **ASSOC**).

If the generated descriptive results use one of Clementine's predefined result formats, they can be displayed using Clementine model browsers. As a result, the results can generate derive nodes, select nodes, and rulesets. The predefined result
formats provide complete compatibility, including the ability to save models and generate new nodes automatically.

CEMI nodes that produce models of this type require only three of the MODEL subsections: PARAMETERS, INPUT_FIELDS, and OUTPUT_FIELDS. All other behavior is determined by the model type rather than the CEMI specification.

The “internal” model formats (used in CEMI modeling nodes with RESULT_FORMAT TREE, RULESET, and ASSOC) now require the result output of the model builder node to use the PMML 2.0 standard XML instead of the previously supported model formats.
This glossary defines terms used in Clementine and data mining in general.

**aggregate.** To combine data across groups. Aggregation is used to create summaries.

**annotation.** Comments associated with a node, model, or stream. These can be added by the user or generated automatically.

**antecedent.** Part of an association rule that specifies a pre-condition for the rule. This is a condition that must be present in a record for the rule to apply to it. The antecedents taken together form the “if” part of the rule. For example, in the rule

milk & cheese => bread

“milk” is an antecedent, and so is “cheese.” *See also consequent.*

**Apriori.** Association rule algorithm, capable of producing rules that describe associations (affinities) between symbolic attributes.

**association.** The extent to which values of one field depend on or are predicted by values of another field.

**balance.** To level the distribution of an attribute (normally symbolic) in a data set by discarding records with common values or duplicating records with rare values.

**batch mode.** The facility to run Clementine without the user interface, so that streams can be run “in the background” or embedded in other applications.

**blanks.** Missing values or values used to indicate missing data.

**Boolean field.** A field that can take only two values, true or false (often encoded as 1 and 0, respectively). *See also flag.*

**boosting.** A technique used by the Build C5.0 node to increase the accuracy of the model. The technique uses multiple models built sequentially. The first model is built normally. The data are then weighted to emphasize the records for which the first model generated errors and the second model is built. The data are then weighted again based on the second model's errors and another model is built, and so on, until
the specified number of models has been built. The boosted model consists of the entire set of models, with final predictions determined by combining the individual model predictions.

**business understanding.** A phase in the CRISP-DM process model. This phase involves determining business objectives, assessing the situation, determining data mining goals, and producing a project plan.

**C&R Trees.** A decision tree algorithm based on minimizing an impurity measure. C&R Trees can handle both symbolic and numeric output fields.

**C5.0.** Rule induction algorithm, capable of producing compact decision trees and rulesets. (The previous version was called C4.5).

**cache.** A store of data associated with a Clementine node.

**case.** A single object or element of interest in the data set. Cases might represent customers, transactions, manufactured parts, or other basic units of analysis. With denormalized data, cases are represented as records in the data set.

**cell.** In a display table, the intersection of one row and one column.

**CEMI (Clementine External Module Interface).** A facility to define Clementine nodes that execute programs external to Clementine.

**chi-square.** A test statistic used to evaluate the association between categorical variables. It is based on differences between predicted frequencies and observed frequencies in a crosstabulation.

**classification.** A process of identifying the group to which an object belongs by examining characteristics of the object. In classification, the groups are defined by some external criterion (contrast with clustering).

**classification and regression trees (C&R Trees).** An algorithm for creating a decision tree based on minimization of impurity measures. Also known as CART.

**classification tree.** A type of decision tree in which the goal of the tree is classification—in other words, a decision tree with a symbolic output field.

**CLEM (Clementine Language for Expression Manipulation).** Language used to test conditions and derive new values in Clementine.

**clustering.** The process of grouping records together based on similarity. In clustering, there is no external criterion for groups (contrast with classification).
**confidence.** An estimate of the accuracy of a prediction. For most models, it is defined as the number of training records for which the model or submodel (such as a specific rule or decision tree branch) makes a correct prediction divided by the number of training records for which the model or submodel makes any prediction.

**connection.** A link between two nodes, along which data records “flow.”

**consequent.** Part of an association rule that specifies the predicted outcome. The consequent forms the “then” part of the rule. For example, in the rule

milk & cheese => bread

“bread” is the consequent. *See also antecedent.*

**correlation.** A statistical measure of the association between two numeric fields. Values range from –1 to +1. A correlation of 0 means that there is no relationship between the two fields.

**CRISP-DM (Cross-Industry Standard Process for Data Mining).** A general process model for data mining. See the *CRISP-DM* manual or CRISP Help for complete information on this process model.

**cross-tabulation.** A table showing counts based on categories of two or more symbolic fields. Each cell of the table indicates how many cases have a specific combination of values for the fields.

**cross-validation.** A technique for testing the generalizability of a model in the absence of a hold-out test sample. Cross-validation works by dividing the training data into $n$ subsets and then building $n$ models with each subset held out in turn. Each of those models is tested on the hold-out sample, and the average accuracy of the models on those hold-out samples is used as an estimate of the accuracy of the model on new data.

**data cleaning.** The process of checking data for errors and correcting those errors whenever possible.

**data mining.** A process for extracting information from large data sets to solve business problems.

**data preparation.** A phase in the CRISP-DM process model. This phase involves selecting, cleaning, constructing, integrating, and formatting data.
**data quality.** The extent to which data have been accurately coded and stored in the database. Factors that adversely affect data quality include missing data, data entry errors, program bugs, etc.

**data set.** A set of data that has been prepared for analysis, usually by denormalizing the data and importing it as a flat file.

**data understanding.** A phase in the CRISP-DM process model. This phase involves collecting initial data, describing data, exploring data, and verifying data quality.

**data visualization.** A process of examining data patterns graphically. Includes use of traditional plots as well as advanced interactive graphics. In many cases, visualization allows you to easily spot patterns that would be difficult to find using other methods.

**data warehouse.** A large database created specifically for decision support throughout the enterprise. It usually consists of data extracted from other company databases. These data have been cleaned and organized for easy access. Often includes a metadata store as well.

**decile.** A division of data into ten ordered groups of equal size. The first decile contains 10% (one-tenth) of the records with the highest values of the ordering attribute.

**decision tree.** A class of data mining models that classifies records based on various field values. The entire sample of cases is split according to a field value, and then each subgroup is split again. The process repeats until further splits cease to improve classification accuracy or until other stopping criteria are met. See also C5.0, C&R Trees, classification tree, and regression tree.

**delimiter.** A character or sequence of characters that appears between fields and/or records in a data file.

**denormalized data.** Data that have been extracted from a relational database (that is, normalized data) and converted to a single table in which each row represents one record and each column represents one field. A file containing denormalized data is called a flat file. This is the type of data typically used in data mining.

**dependent variable.** A variable (field) whose value is assumed to depend on the values of other variables (fields). Also known as an output field or variable.
**deployment.** A phase in the CRISP-DM process model. This phase involves plan deployment, monitoring and maintenance, producing a final report, and reviewing the project.

**derived field.** A field that is calculated or inferred from other fields. For example, if you have share price and earnings per share for stocks in your database, you could divide the former by the latter to get the P/E ratio, a derived field.

**diagram.** The current contents of the stream canvas. May contain zero or one or more valid streams.

**directed web.** A display used for examining the relations between symbolic data fields and a target symbolic data field.

**direction.** Whether a field will be used as an input, output, or both or will be ignored by modeling algorithms.

**distribution.** A characteristic of a field defined by the pattern of values observed in the data for that field.

**domain knowledge.** Knowledge and expertise that you possess related to the substantive business problem under consideration, as distinguished from knowledge of data mining techniques.

**downstream.** The direction in which data is flowing; the part of the stream after the current node.

**equation.** Numeric model based on linear regression, produced by a regression node.

**evaluation.** A phase in the CRISP-DM process model. This phase involves evaluating results, reviewing the data mining process, and determining the next steps.

**factor analysis.** A method of data reduction that works by summarizing the common variance in a large number of related fields using a small number of derived fields that capture the structure in the original fields. See also PCA.

**feature.** An attribute of a case or record. In database terms, it is synonymous with field. See also field, variable.

**field.** A datum associated with a record in a database. A measured characteristic of the object represented by the record. See also feature, variable.

**filler.** Operation to replace values in a record, often used to fill blanks with a specified value.
filter. Discard fields from a record.

fixed file. A file whose records are of constant length (number of characters). Fields are defined by their starting position in the record and their length.

flag. A symbolic field with exactly two valid values, usually some variation of true and false.

flat file. A data set represented by a single table with a row for each record and a column for each field. Composed of denormalized data.

generated model. An icon on the Models tab in the managers window, representing a model generated by a modeling node.

global values. Values associated with a whole data set rather than with individual records.

GRI (generalized rule induction). An association rule algorithm capable of producing rules that describe associations (affinities) between attributes of a symbolic target.

histogram. A graphical display of the distribution of values for a numeric field. It is created by dividing the range of possible values into subranges, or bins, and plotting a bar for each bin indicating the number of cases having a value within the range of the bin.

history. Operation to integrate values from a sequence of previous records into the current record.

impurity. An index of how much variability exists in a subgroup or segment of data. A low impurity index indicates a homogeneous group, where most members of the group have similar values for the criterion or target field.

input field. A field used to predict the values of one or more output fields by a machine learning technique. See also predictor.

instantiate. To specify the valid values of a field. Fields can be partially instantiated. For example, a field can be defined as a set field, but the specific members of the set that define valid values may be left undefined. Fields can also be fully instantiated, where all the necessary information is defined for the field. Instantiation is typically performed automatically by passing the data through a Type node, but you can also define or edit instantiation information manually in the Type node.

integer. A number with no decimal point or fractional part.
**interaction.** In a statistical model, an interaction is a type of effect involving two or more fields (variables) in which the effect of one field in predicting the output field depends on the level of the other input field(s). For example, if you are predicting response to a marketing campaign, you may find that high price leads to decreased response for low-income people but increased response for high-income people.

**iterative.** Involving repeated applications of a step or a series of steps. Counting is a simple iterative procedure, which works by taking the step “add one to the previous value” and applying it repeatedly. An iteration is a single pass through the steps of an iterative process.

**k-means.** An approach to clustering that defines \( k \) clusters and iteratively assigns records to clusters based on distances from the mean of each cluster until a stable solution is found.

**Kohonen network.** A type of neural network used for clustering. Also known as a self-organizing map (SOM).

**lift.** Improvement in expected return caused by the use of a classifier or model over that expected with no classification or prediction. The higher the lift, the better the classifier or model.

**linear regression.** A mathematical technique for estimating a linear model for a continuous output field.

**logistic regression.** A special type of regression model used when the output field is symbolic.

**machine learning.** A set of methods for allowing a computer to learn a specific task—usually decision making, estimation, classification, prediction, etc.—without having to be (manually) programmed to do so. Also, the process of applying such methods to data.

**main effect.** In a statistical model, a main effect is the direct effect of an input field (predictor) on the output field (target), independent of the values of other input fields. Contrast with interaction.

**market basket analysis.** An application of association-based models that attempts to describe pairs or clusters of items that tend to be purchased by the same customer at the same time.

**matrix.** A matrix-style or cross-tabulation display format.
**mean.** The average value for a field (variable). The mean is a measure of the center of the distribution for a field. Compare with **median** and **mode**.

**median.** The value for a field below which 50% of the observed values fall; the value that splits the data into an upper half and a lower half. The median is a measure of the center of the distribution for a field. Compare with **mean** and **mode**.

**merge.** To combine multiple tables into a single table by joining pairs (or n-pairs) of records together.

**metadata.** Literally, data about data. Metadata is information about the data in your data store. It typically contains descriptions of fields, records, and relationships between fields, as well as information about how the data store was assembled and how it is maintained.

**misclassification matrix.** A crosstabulation of predicted values versus observed values for a given classification model. Shows the different types of errors made by the model. Sometimes called a **confusion matrix**.

**mode.** The most frequently observed value for a field. The mode is useful for summarizing symbolic fields. Compare with **mean** and **median**.

**model.** A mathematical equation that describes the relationship among a set of fields. Models are usually based on statistical methods and involve assumptions about the distributions of the fields used in the model, as well as the mathematical form of the relationship.

**modeling.** A phase in the CRISP-DM process model. This phase involves selecting modeling techniques, generating test designs, and building and assessing models.

**multilayer perceptron (MLP).** A common type of neural network, used for classification or prediction. Also called a back propagation network.

**multiplot.** A graph on which several fields are plotted at once.

**neural network.** A mathematical model for predicting or classifying cases using a complex mathematical scheme that simulates an abstract version of brain cells. A neural network is trained by presenting it with a large number of observed cases, one at a time, and allowing it to update itself repeatedly until it learns the task.

**node.** A processing operation in Clementine's visual programming environment. Data flows from, into, or through a node.
nominal regression. See logistic regression.

normalized data. Data that have been broken into logical pieces that are stored separately to minimize redundancy. For example, information about specific products may be separated from order information. By doing this, the details of each product appear only once, in a products table, instead of being repeated for each transaction involving that product. Normalized data are usually stored in a relational database, with relations defining how records in different tables refer to one another. Contrast with denormalized data.

ODBC (open database connectivity). ODBC is a data exchange interface, allowing programs of various types to exchange data with each other. For example, if your database system is ODBC-compliant, the task of transferring data to and from the database is made much simpler.

outlier. A record with extreme values for one or more fields. Various technical definitions are used for determining which specific cases are outliers. The most common criterion is that any case with a value greater than three standard deviations from the mean (in either direction) is considered an outlier.

output field. A field to be predicted by a machine-learning technique. See also target and dependent variable.

overfitting. A potential problem with model estimation in which the model is influenced by some quirks of the data sample. Ideally, the model encodes only the true patterns of interest. However, sometimes data mining methods can learn details of the training data that are not part of a general pattern, which leads to models that don't generalize well. Cross-validation is a method for detecting overfitting in a model.

palette. A collection of node icons from which new components can be selected.

parameter. A value used like a variable for modifying the behavior of a stream without editing it by hand.

PCA (principal components analysis). A method of data reduction that works by summarizing the total variance in a large number of related fields using a small number of derived fields. See also factor analysis.

prediction. An estimate of the value of some output field for an unknown case, based on a model and the values of other fields for that case.
**predictor.** A field in the data set that is used in a model or classifier to predict the value of some other field (the output field). *See also input field.*

**probability.** A measure of the likelihood that an event will occur. Probability values range from 0 to 1; 0 implies that the event never occurs, and 1 implies that the event always occurs. A probability of 0.5 indicates that the event has an even chance of occurring or not occurring.

**project tool.** Clementine's facility for organizing and managing the materials associated with a data mining project (streams, graphs, and documents). Includes the Report manager.

**pruning.** Reducing the size of a model to improve its generalizability and, in some cases, its accuracy. With rule induction, this is achieved by removing the less significant parts of the decision tree. With neural networks, underused neurons are removed.

**quantile.** Division of data into ordered groups of equal size. Examples of quantiles are quartiles, quintiles, and deciles.

**quartile.** A division of data into four ordered groups of equal size. The first quartile contains 25% (one-fourth) of records with the highest values of the ordering attribute.

**query.** A formal specification of data to be extracted from a database, data warehouse, or data mart. Queries are often expressed in structured query language (SQL). For example, to analyze records for only your male customers, you would make a query on the database for all records in which customer's gender has the value *male*, and then analyze the resulting subset of the data.

**quintile.** A division of data into five ordered groups of equal size. The first quintile contains 20% (one-fifth) of the records, with the highest values of the ordering attribute.

**RBFN (radial basis function network).** A type of neural network used for predictive modelling but internally based on clustering.

**real number.** A number with a decimal point.

**record.** A row in a database; for denormalized data, synonymous with *case*.

**refined model.** A model that is executable and can be placed in streams and used to generate predictions. Most modeling nodes produce refined models. Exceptions are GRI and Apriori, which produce unrefined models.
regression tree. A tree-based algorithm that splits a sample of cases repeatedly to derive homogeneous subsets, based on values of a numeric output field.

relational database. A data store designed for normalized data. A relational database usually consists of a set of tables and a set of relations that define how records from one table are related to records from other tables. For example, a product ID may be used to link records in a transaction table with records in a product detail table.

Report Manager. Clementine's facility for automatically producing draft project reports. The Report Manager is part of the projects window.

rough diamond. See unrefined model.

row. A record, or case, in a database.

rule induction. The process of automatically deriving decision-making rules from example cases.

ruleset. A decision tree expressed as a set of independent rules.

sample. A subset of cases selected from a larger set of possible cases (called the population). The data you analyze are based on a sample; the conclusions you draw are usually applied to the larger population. Also, to select such a subset of cases.

scatterplot. A data graph that plots two (or sometimes three) numeric fields against each other for a set of records. Each point in the scatterplot represents one record. Relationships between fields can often be readily seen in an appropriate scatterplot.

scoring. The process of producing a classification or prediction for a new, untested case. An example is credit scoring, where a credit application is rated for risk based on various aspects of the applicant and the loan in question.

script. In Clementine, a series of statements or commands that manipulate a stream. Scripts are used to control stream execution and automate data mining tasks.

segment. A group or subgroup having some set of properties in common. Usually used in a marketing context to describe homogeneous subsets of the population of potential customers.

segmentation. A process of identifying groups of records with similar values for a target field. The process takes the whole set of records and divides them into subgroups, or segments, based on characteristics of the records.

select. Extract a subset of data records based on a test condition.
sensitivity analysis. A technique for judging the relevance of data fields to a neural network by examining how changes in input affect the output.

sequence. The ordering of records.

set field. A symbolic field with more than two valid values.

significance (statistical). A statement regarding the probability that an observed difference is attributable to random fluctuations (that is, attributable to chance). The smaller this probability is, the more confident you can be that the difference represents a true difference.

slot parameter. A setting in a Clementine node that can be treated like a parameter and set in a script, using a parameter-setting dialog box or the Clementine command line. Also called node or stream properties.

SQL (structured query language). A specialized language for selecting data from a database. This is the standard way of expressing data queries for most database management systems.

standard deviation. A measure of the variability in the values of a field. It is calculated by taking the difference between each value and the overall mean, squaring it, summing across all of the values, dividing by the number of records (or sometimes by the number of records minus one), and then taking the square root. The standard deviation is equal to the square root of the variance.

statistics. Generally, a set of methods used to derive general information from specific data. The term is also used to describe the computed values derived from these methods.

stream. A path of connected nodes along which data flows.

string. A piece of text made up of a sequence of characters—fred, Class 2, or 1234, for example.

supervised learning. A learning task where there is an output field with observed data that can be used to train a learning algorithm. The algorithm attempts to build a model that produces predictions that match the observed output values as closely as possible. This external criterion of observed output values is said to supervise the learning process. Compare to unsupervised learning.
support. For an association or sequence rule, a measure of the rule's prevalence in the training data or the proportion of the training data to which the rule applies. It is defined differently for association rules and for sequences. For association rules, it is the proportion of training records for which the antecedents of the rule are true (sometimes expressed as a percentage). For sequences, it is the proportion of training IDs that contain at least one instance of the entire sequence, including the consequent.

symbolic field. A field whose values are restricted to a particular list of valid values, usually representing categories. Symbolic field values are not treated as mathematical numbers, even when coded with numeric values. For example, you cannot multiply or divide symbolic field values. Flags and set fields are examples of symbolic fields.

target. The field that you want to predict, whose value is assumed to be related to the values of other fields (the predictors). Also known as an output field or dependent variable.

time series analysis. Data analysis techniques in which measurements are taken on the same unit at several points in time. Also, the application of these techniques.

transformation. A formula applied to values of a field to alter the distribution of values. Some statistical methods require that fields have a particular distribution. When a field's distribution differs from what is required, a transformation (such as taking logarithms of values) can often remedy the problem.

two-step clustering. A clustering method that involves preclustering the records into a large number of subclusters and then applying a hierarchical clustering technique to those subclusters to define the final clusters.

type. Definition of the valid values that a field can have.

unrefined model. A model that is not executable but that could potentially be transformed into a useful executable model. The GRI and Apriori nodes both produce these.

unsupervised learning. A learning task lacking an external criterion for testing output values. The learning algorithm must impose its own structure on the problem to derive a solution. Clustering models are examples of unsupervised learning. Compare to supervised learning.

upstream. The direction from which data has come; the part of the stream preceding the current node.
user input. Interactive specification of a data set by the user—for example, for purposes of testing a model.

variable. In general, any measured characteristic that can vary across records. Variables are represented as fields in a database; for most purposes, variable, attribute, and field are synonymous.

variable file. A file whose records are of different lengths (number of characters) but have a constant number of fields that are separated by delimiters.

variance. A measure of the variability in the values of a field. It is calculated by taking the difference between each value and the overall mean, squaring it, summing across all of the values, and dividing by the number of records (or sometimes by the number of records minus one). The variance is equal to the square of the standard deviation.

vingtile. A division of data into 20 ordered groups of equal size. The first vingtile contains 5% (one-twentieth) of the records, with the highest values of the ordering attribute.

visual programming. Specifying how to manipulate and process a sequence of data records by positioning and editing graphical objects.

web. A display used for examining the relations between symbolic data fields.
1-in-n sampling, 128
3-D graphs, 234
abs, 674
absolute confidence difference to prior (apriori evaluation measure), 344
adding
records, 131
to a project, 579
additional information panel
decision tree node, 422
advanced output
factor/PCA node, 403, 406
linear regression node, 338, 401
logistic regression node, 356, 397
aggregate node
overview, 131
setting options, 132
aggregating data, 841
aggregating records, 198
Akaike Information Criterion
linear regression, 338
algorithms, 7, 382
allbutfirst, 679
allbutlast, 679
alpha
neural net node, 323
alphabefore, 673, 679
Amemiya Prediction Criterion
linear regression, 338
analysis browser
edit menu, 489
file menu, 489
interpreting, 506
analysis node, 504
analysis tab, 504
output tab, 491
animation, 232, 236
annotating, 841
nodes, 70
streams, 83
annotations
folder, 583
project, 583
AnswerTree
launching from Clementine, 541, 551
antecedent, 841
anti-join, 135
append node
field matching, 144
overview, 143
setting options, 144
tagging fields, 141
applications of data mining, 50
Apriori, 841
apriori node, 341, 342, 343
evaluation measures, 343
expert options, 343
options, 342
tabular vs. transactional data, 310
arccos, 676
arcosh, 676
arcsin, 676
arcsinh, 676
arctan, 676
arctan2, 676
arctanh, 676
arguments
command file, 732
server connection, 732
system, 734
ascending order, 134
assigning data types, 147, 148
association, 841
association plots, 276, 278
association rules, 45, 339, 341, 407, 408, 409, 411, 412, 414, 415, 417, 454, 455, 456, 459
for sequences, 373
asymptotic correlations
logistic regression, 356, 397
asymptotic covariance
logistic regression, 356
attribute, 41
audit
data audit node, 510
initial data audit, 510
automatic recode, 184, 184, 188
automation, 205

Index
auto-typing, 151, 154
balance, 841
balance factors, 130
balance node
  generating, 261, 266, 274
  overview, 129
  setting options, 130
baseline
  evaluation chart options, 296
batch mode, 39, 205, 597, 729, 841
  invoking software, 592, 731
  log, 593
  log files, 735
  output, 596
  parameters, 594
  scheduling streams, 591
  scripting, 594, 606
  using arguments, 592, 731
best line
  evaluation chart options, 296
biased data, 129
binning node
  count-based bins, 192
  fixed-width bins, 190
  mean/standard deviation bins, 195
  overview, 188
  ranks, 194
  setting options, 189
  viewing bins, 196
bitwise functions, 677
blank handling, 148, 155
  binning node, 189
  CLEM functions, 695
  filling values, 181
blanks, 218, 223, 224, 225, 522, 841
blank values
  quality node, 523
Boolean field, 841
boosting, 330, 427, 841
build rule node, 367, 418
  loading, 88
bulk loading, 536
business rule
  evaluation chart options, 297
  business understanding, 841
C&R tree node, 367, 368, 369, 418, 419
  case weights, 309
  frequency weights, 309
  impurity measures, 369
  misclassification costs, 333
  prior probabilities, 369, 372
  pruning, 369
  stopping criteria, 369
  stopping options, 371
  surrogates, 369
  tree depth, 368
C&R trees, 384, 424, 841
C5.0, 412, 414, 415, 417, 418, 419, 424, 427, 841
  boosting, 427
C5.0 node, 329, 330
  boosting, 330
  misclassification costs, 330, 333
  options, 330
  pruning, 330
cache, 841
  enabling, 32, 68
  flushing, 68, 73
  options for nodes, 68
  saving, 68
  setting up a cache, 66
  supernodes, 570
cache file node, 111
  loading, 88
cache statements, 717
case, 41, 841
C code, 464, 465, 480
cell, 841
CEMI, 607, 841
  custom node icons, 627, 629
  examples, 616
  loading new nodes, 624, 625
  removing nodes, 624, 626
  restrictions, 615
  specification file, 607, 609, 799
  system architecture, 608
  writing programs, 630
CHAID, 384
characters, 663, 665
checking CLEM expressions, 215
checking types, 160
classes, 575, 578
classification, 841
classification and regression trees, 841
classification table
  logistic regression, 356
classification, 841
classification and regression trees, 841
clear values, 119
CLEM, 1, 210, 217, 704, 841
  building expressions, 212
  checking expressions, 215
datatypes, 664, 664, 665, 665, 665, 666
  examples, 208
  expressions, 207, 663, 669
  function categories, 216
  functions, 212
  introduction, 39, 205
  language, 663
  missing values, 228
  scripting, 597, 697
Clementine, 24
  features at a glance, 24
  getting started, 21
  installing, 18, 19
  options, 31
  overview, 1, 17
  running from command line, 592, 592, 731, 731
  uninstalling, 21
Clementine Application Templates (CATs)
  data mapping tool, 89
Clementine Batch, 731
Clementine Solution Publisher, 531
CLEM expressions, 125
CLEM functions
  bitwise, 677
  blanks and nulls, 695
  comparison, 673
  conversion, 672
datetime, 684
  global, 694
  information, 672
  list of available, 670
  logical, 674
  numeric, 674
  random, 678
  sequence, 688
  special functions, 696
  string, 679
  trigonometric, 676
Cleo, 10
  stream prerequisites, 485
  wizard, 484, 485
client
  default directory, 32
close statement, 713
cluster analysis
  number of clusters, 366
clustering, 48, 324, 345, 364, 428, 447, 447, 449, 451, 452, 841
  overall display, 428
  viewing clusters, 428
cluster viewer
  display options, 436, 438, 439
  importance, 440
  interpreting results, 430
  overview, 428
  text view, 441
  using, 435
  view all, 436, 438
coercing values, 160
coincidence matrix
  analysis node, 504
collection node
  creating, 274
  graph window, 274
  overview, 272
collinearity diagnostics
  linear regression, 338, 401
colors
  setting, 36
column order
  table browser, 496
column width
  table node, 494
column-wise binding, 536
combining data, 143
combining data from multiple files, 135
comma, 73
command line, 21
  list of arguments, 732, 734, 735
  multiple arguments, 732
  parameters, 735
  running Clementine, 592, 592, 731, 731
comment characters
  in variable files, 99
comments, 698
commit size, 536
comparison functions, 673
concatenating records, 143
condition monitoring, 631
conditions, 207
  specifying a series, 178
confidence, 841
  apriori node, 342
  decision tree node, 422
  for sequences, 456
  GRI node, 340
  neural networks, 388
  sequence node, 376
confidence difference (apriori evaluation measure), 344
confidence intervals
  linear regression, 338, 401
  logistic regression, 356
confidence ratio (apriori evaluation measure), 344
connecting nodes
  scripting, 720
connections, 841
  database, 108
  server, 22
connect statement, 720
consequent, 841
content field(s)
  sequence node, 374
contiguous data sampling, 128
contiguous keys, 132
continuations, 698
CONTROLS subsection, 619, 814
  guidelines, 611
  SHEET subsection, 816
  TAB subsection, 818
conventions, 671
convergence options
  logistic regression node, 355
conversion functions, 672
converting sets to flags, 197
copy, 26
copying type attributes, 163
CORE specification, 609, 617, 804
  CONTROLS subsection, 611, 619, 814
  EXECUTE subsection, 610, 618, 810
  INPUT_DATA subsection, 614, 826
  INPUT_FIELDS subsection, 613, 819
  MAPPING_FILE subsection, 614, 828
  MODEL_FILES subsection, 835
  OPTIONS subsection, 610, 619, 812
  OUTPUT_DATA subsection, 614, 621, 826
  OUTPUT_FIELDS subsection, 613, 620, 821
  PARAMETERS subsection, 610, 617, 805
  RESULTS subsection, 615, 833
  RETURN_CODE subsection, 615, 621, 832
correlations, 518, 841
cos, 676
cosh, 676
counts, 192
  statistics output, 520
covariance matrix
  linear regression, 338, 401
creating
  new fields, 167, 169
  synthetic data, 114
CRISP-DM, 1, 575
  data understanding, 97
  projects view, 576
CRISP-DM process model, 52, 53, 841
  data preparation, 147
cross-tabulation, 841
cross-validation, 841
cut, 26
cut points
  binning node, 188
data
  aggregating, 131
  audit, 510
  availability, 50
  balancing, 55
  coverage, 50
  expertise, 51
  exploring, 510
  size of, 51
storage, 104, 154, 181
storage type, 155
data access pack, 18
data audit browser
edit menu, 514
file menu, 514
generating graphs, 516
generating nodes, 516
data audit node, 510
sample tab, 513
settings tab, 510
use in data mining, 50
use in exploration, 41
database
bulk loading, 536
connecting to data, 108
reading data from, 106, 106
selecting a table, 109
database node
query editor, 110
database output node, 533
data source, 534
export tab, 534
schema, 535
table name, 534
data cleaning, 843
data mapping tool, 89, 90
data mining, 41, 843
choosing a modeling method, 55
strategy, 52
tips, 55
data preparation, 125, 843
data quality, 522, 843
data reduction, 359
data set, 843
data storage, 183
data streams
building, 57, 58
data types, 101, 147, 148, 206
instantiation, 152
data understanding, 125, 843
data visualization, 843
data warehouse, 843
date_before, 673
date/time, 150, 220
date formats, 666, 667
date functions, 666, 667
@TODAY, 684
date_before, 684
date_days_difference, 684
date_in_days, 684
date_in_months, 684
date_in_weeks, 684
date_in_years, 684
date_months_difference, 684
date_weeks_difference, 684
date_years_difference, 684
dates, 217
setting formats, 73
datetime functions
datetime_date, 684, 684
datetime_day, 684
datetime_day_name, 684
datetime_day_short_name, 684
datetime_hour, 684
datetime_in_seconds, 684
datetime_minute, 684
datetime_month, 684
datetime_month_name, 684
datetime_month_short_name, 684
datetime_now, 684
datetime_second, 684
datetime_time, 684
datetime_timestamp, 684
datetime_weekday, 684
datetime_year, 684
decile, 843
decimal, 73
decimal symbol, 99
Excel node, 545
flat file output node, 539
decision tree node, 418, 419, 423, 424, 426
additional information panel, 422
rule frequencies, 422
surrogates, 422
decision trees, 43, 329, 367, 418, 419, 424, 843
decreasing data, 126, 127
default
project phase, 577
delete statements, 717
deleting
output objects, 488
duplicate fields, 135, 165
duplicate records, 145
duplicate statements, 717
Durbin-Watson test
  linear regression, 338, 401
ingesting dialog box, 611
  enabling controls, 816
  layout, 815
  sheets, 816
  tabs, 818
eigenvalues
  factor/PCA node, 363
empty strings
  quality node, 523
encapsulating nodes, 557
endif statements, 727
EOL characters, 99
equal counts, 192
equamax rotation
  factor/PCA node, 363
equation, 843
error codes, 480
error messages, 78
essential fields, 89, 93
eta
  neural net node, 323
evaluating models, 504
evaluation, 843
evaluation chart node
  business rule, 297
creating, 296, 297
  hiding lines, 304
  hit condition, 297
  overview, 290
  reading results, 299
  score expression, 297
  using the graph, 301
evaluation measures
  apriori node, 343
examples
  condition monitoring, 631
  fraud detection, 638
  market basket analysis, 652
  overview, 631
  retail analysis, 647
Index

Excel, 106, 545
  launching from Clementine, 551
Excel node, 545
exceptions, 56
execute statement, 711
EXECUTE subsection, 610, 618, 810
  command line arguments, 812
  conditional execution, 811
  server mode, 812
executing scripts, 727, 729
execution
  for streams, 85
  specifying the order of, 572
exhaustive pruning
  neural net node, 312
exit, 727
exit commands, 727
exp, 674
expert options
  apriori node, 343
  C&R tree node, 369
  factor/PCA node, 361
  k-means node, 347
  Kohonen node, 328
  logistic regression node, 353
  neural net node, 318, 319, 321, 322, 323
  sequence node, 377
expert output
  factor/PCA node, 362
  linear regression node, 337, 338
  logistic regression node, 356
exploring data
  data audit node, 510
exporting
  C code, 465
  error codes, 480
  field names, 480
  generated models, 382
  k-means models, 477, 478, 478, 479, 479
  Kohonen nets, 474, 474, 475, 475, 476
  models, 464, 480
  neural nets, 467, 467, 468, 468, 469
  output, 489
  PMML, 465
  radial basis function nets, 470, 471, 471, 472, 472
  rules, 466, 466, 466
  supernodes, 573
  to Cleo, 484
  to PredictiveMarketing, 481
exporting data
  flat file format, 539
  SAS format, 544
  to a database, 533
  to AnswerTree, 540
  to Excel, 545, 545
  to SPSS, 540
export statements, 722
Expression Builder, 1, 125
  accessing, 211
  overview, 210
  using, 212
expressions, 663
extension
  derived field, 170
factor/PCA node, 359, 360, 361
  eigenvalues, 363
  estimation methods, 360
  expert options, 361
  expert output, 362
  factor scores, 361
  iterations, 361
  missing value handling, 361
  number of factors, 361
  rotation options, 363
factor analysis, 359, 403, 404, 406, 843
factor equation node, 403, 404, 406
false values, 159
feature, 843
feedback graph
  neural net node, 315
  field, 483
field attributes, 163
field derivation formula, 173
field names, 167
  data export, 533, 539, 541, 544
  exporting, 480
field operations nodes, 147
  binning node, 188
  derive node, 167
  field reorder node, 201
filler node, 181
filter node, 164
generating from a data audit, 516
history node, 199
reclassify node, 184
scripting properties, 751
type node, 148
field ops palette, 802
field options
modeling nodes, 308
field parameters, 807, 815
field reorder node, 201
automatic sorting, 204
custom ordering, 202
setting options, 202
fields, 41, 663, 666
  deriving multiple fields, 170
  in CLEM expressions, 213
  reordering, 201
  selecting multiple, 172
  turning off in scripting, 758
  viewing values, 214
field selection
  linear regression node, 334
field types, 148
filler node, 843
  missing values, 227
  overview, 181
filtering fields, 139, 164, 843
  for SPSS, 542
filter node
  generating from neural network model, 392
  overview, 164
  setting options, 165
fixed-field text data, 101
fixed file node
  overview, 101
  setting options, 101
fixed files, 843
flag parameters, 806, 815
flags, 843
  combining multiple flags, 732
  command line arguments, 592, 731
flag type, 150, 151, 159
flat file, 843
flat file output node, 539
  export tab, 539
fonts, 36
format files, 114
fracof, 674
fractional ranks, 194
fraud detection, 638
freefield text data, 98
frequencies, 192
decision tree node, 422
functions, 666, 667, 671
  @AVE, 696
  @BLANK, 227, 228, 672, 695
  @DIFF, 688
  @FIELD, 210, 210, 228, 696
  @GLOBAL_AVE, 696
  @GLOBAL_MAX, 694
  @GLOBAL_MEAN, 694
  @GLOBAL_MIN, 694
  @GLOBAL_SDEV, 694
  @GLOBAL_SUM, 694
  @INDEX, 688
  @LAST_NON_BLANK, 688, 695
  @LAST_NON_BLANK, 688, 695
  @MAX, 688
  @MEAN, 688
  @MIN, 688
  @NULL, 228, 672, 695
  @OFFSET, 688
  @PREDICTED, 210, 210, 696
  @SDEV, 688
  @SINCE, 688
  @SUM, 688
  @TARGET, 210, 210, 696
  @THIS, 688
  @TODAY, 684
  examples, 208
  in CLEM expressions, 212
gains charts, 290, 299
generated k-means node, 447, 447, 449
generated Kohonen node, 443, 444, 445
generated models, 381, 385, 388, 389, 392, 393, 395, 397, 398, 399, 401, 403, 404, 405, 406, 407, 408, 409, 412, 414, 415, 417, 418, 419,
Index

exporting, 382, 386
generating processing nodes from, 385
menus, 386
node properties, 780
printing, 386
saving, 386
saving and loading, 382
scoring data with, 385
tabs, 386
using in streams, 385
generated models palette, 381, 722
saving and loading, 382
generated net node, 388, 389
generated sequence ruleset, 411
generated sequence rules node, 454, 455, 456, 459
generated TwoStep cluster node, 450, 451, 452
generating flags, 198
global functions, 694
global parameters, 701, 702
global values, 529, 846
goodness-of-fit chi-square statistics
logistic regression, 356, 397
graph nodes, 231, 240
scripting properties, 758
graphs
3-D, 234
adding to projects, 578
animation, 232
appearance options, 237
collections, 272
color overlay, 232
distributions, 259
editing regions, 256
evaluation charts, 290
exporting, 242
generating from a data audit, 516
histograms, 264
multiplot, 256
output, 236
panel overlay, 232
plots, 243
printing, 242
saving, 242
shape overlay, 232
size overlay, 232
transparency, 232
webs, 276, 278
with animation, 236
graphs palette, 802
GRI node, 339, 340, 846
options, 340
grouping
values, 261
handling missing values, 147
hasendstring, 679
HASH_DATA keyword, 803
hasmidstring, 679
hasstartstring, 679
hassubstring, 679, 679
help
accessing, 30
types of, 30
helper applications, 550
histogram, 846
histogram node
creating, 265
overview, 264
using the graph, 266
history, 846
decision tree node, 422
history node, 200
overview, 199
hits
evaluation chart options, 297
hot keys, 28
HTML
report node, 492
icons
setting options, 76
ICON subsection, 803
ID field
sequence node, 374
id parameters, 810
if, then, else functions, 674
if, then, else statements, 179
if statements, 727
imbalanced data, 129
importance
in the cluster viewer, 440
Index

importing
  PMML models, 382
  supernodes, 573
impurity, 846
incomplete records, 137
increasing performance, 127
information difference (apriori evaluation measure), 344
information functions, 672
inner join, 135
INPUT_DATA subsection, 614, 826
INPUT_FIELDS subsection, 613, 819
input field, 846
insert statements, 722
installation, 18
  CEMI nodes, 625
instances
  decision tree node, 422
instantiation, 148, 150, 151, 152, 154, 846
  source node, 121
  type node, 153
integer_bitcount, 677
integer_leastbit, 677
integer_length, 677
integer functions, 696
integer ranges, 157
integers, 663, 664, 846
interaction, 846
interrupting scripts, 729
intof, 674
introduction, 663
  Clementine, 1, 17
invalid values, 522
is functions
  is_date, 672
  is_datetime, 672
  is_integer, 672
  is_number, 672
  is_real, 672
  is_string, 672
  is_time, 672
  is_timestamp, 672
isalphacode, 679
isendstring, 679
islowercode, 679
ismidstring, 679
isnumbercode, 679
isstartstring, 679
issubstring, 679, 679
issubstring_lim, 679
isuppercode, 679
iteration history
  logistic regression, 356
iterative, 846
jittering, 249
joining data sets, 143
joins, 135, 135, 137
  partial outer, 139
justification
  table node, 494
key fields, 132, 198
key method, 135
key value for aggregation, 132
k-means clustering, 48, 345, 447, 449, 847
k-means generated models, 447
k-means models
  exporting, 477, 478, 478, 479, 479
k-means node, 345, 346, 347
  distance field, 346
  encoding value for sets, 347
  expert options, 347
  large sets, 73
  stopping criteria, 347
knowledge discovery, 41
Kohonen generated models, 443
Kohonen nets
  exporting, 474, 474, 475, 475, 476
Kohonen networks, 44, 48, 847
Kohonen node, 324, 326, 328
  binary set encoding option (removed), 326
  expert options, 328
  feedback graph, 326
  learning rate, 328
  neighborhood, 324, 328
  stopping criteria, 326
labels
  value, 384
  variable, 384
lagged data, 199
language
  options, 31
large databases, 125
  performing a data audit, 510
learning rate
  neural net node, 323
length, 679
lift, 408, 847
lift charts, 290, 299
likelihood ratio test
  logistic regression, 356, 397
linear regression, 47, 334, 399, 401, 405, 847
linear regression equation node, 399, 401, 405
linear regression node, 334, 334, 336
  advanced (expert) output, 338
  backwards estimation method, 334
  expert options, 336
  expert output, 336
  field selection, 334
  forwards estimation method, 334
  missing value handling, 336
  stepping criteria (field selection), 336, 337
  stepwise estimation method, 334
  weighted least squares, 309
line plots, 231, 243, 256
links
  web node, 280
list parameters
  modifying in scripts, 699
lists, 663, 665
loading
  generated models, 382
  nodes, 88
  states, 88
locale
  options, 31
local variables, 701, 701
locchar, 679
locchar_back, 679
log, 674
log10, 674
log files, 593
  arguments, 735
logical functions, 674
logistic regression, 47, 348, 384, 393, 395, 397, 847
logistic regression equation node, 392, 395, 397
  equations, 393
  model tab, 393
logistic regression node, 348, 349, 353
  advanced (expert) output, 356
  convergence criteria, 353
  convergence options, 355
  expert options, 353
  expert output, 353
  model form, 349
  stepping criteria (field selection), 357
log-odds, 393
lowertoupper, 679
machine learning, 41, 42, 847
main data set, 144
main effect, 847
Mallows’ Prediction Criterion
  linear regression, 338
managers
  models tab, 382
  outputs tab, 488
mandatory fields, 94
MAPPING_FILE subsection, 614, 828
  MAP_TYPE subsection, 829
  MAPPING_FORMAT subsection, 831
mapping data, 93
mapping fields, 89
market basket analysis, 652, 847
matrix, 847
matrix browser
  edit menu, 489
  file menu, 489
  generate menu, 502
matrix node, 498
  appearance tab, 500
  column percentages, 500
  cross-tabulation, 500
  highlighting, 500
  output browser, 502
  output tab, 491
  row percentages, 500
  settings tab, 498
  sorting rows and columns, 500
max, 673
Index

maximum
  set globals node, 530
  statistics output, 520
maximum value for aggregation, 132
mean, 847
  binning node, 195
  set globals node, 530
  statistics output, 520
mean/standard deviation
  used to bin fields, 195
mean value for aggregation, 132
mean value for records, 131
median, 847
  statistics output, 520
member, 673
member (SAS import)
  setting, 114
memory
  managing, 31, 32
merge node
  filtering fields, 139
  overview, 135
  setting options, 137
  tagging fields, 141
merging records, 847
metadata, 51, 847
middle mouse button
  simulating, 28, 61
min, 673
minimizing, 27
minimum
  set globals node, 530
  statistics output, 520
minimum value for aggregation, 132
misclassification costs, 333
  C5.0 node, 330
misclassification matrix, 847
missing values, 147, 155, 218, 224, 225, 227, 522
  CLEM expressions, 228
  filling, 223
  handling, 223
  in Aggregate nodes, 131
  in fields, 226
  in records, 226
mode, 847
  statistics output, 520
MODEL_FILES subsection, 835
model evaluation, 290
model fit
  linear regression, 338, 401
model fitting information
  logistic regression, 397
modeling, 847
modeling nodes, 307, 311, 324, 329, 334, 339, 341, 345, 348, 359, 364, 367, 373, 608, 802, 803, 805
  scripting properties, 767
modeling palette, 803
models
  adding to projects, 578
  duplicate names, 722
  exporting, 464, 480
  importing, 382
  scripting, 722, 722, 723, 723
MODEL specification, 610, 836
  for refined models, 837
  for unrefined models, 837
models tab
  saving and loading, 382
modifying data values, 167
momentum
  neural net node, 323
mouse
  using in Clementine, 28, 61
multi-layer perceptrons, 311, 847
multiple derive, 170
multiple fields
  selecting, 172
multiple inputs, 135
multiple regression, 334
multiplot node, 847
  creating, 257
  overview, 256
  using the graph, 259
multiset command, 738
NAME keyword, 801
natural order
  altering, 201
negate, 674
network
  web graph, 280
neural net node, 311, 312, 317, 318, 319, 321, 322, 323
  alpha, 323
dynamic training method, 312, 318
  eta, 323
exhaustive prune training method, 312, 323
expert options, 317, 318, 321, 322, 323
feedback graph, 315
field options, 308
large sets, 73
learning rate (eta), 323
momentum (alpha), 323
multiple training method, 312, 319
persistence, 317, 319, 321, 322
prune training method, 312, 321
quick training method, 312, 317
radial basis function network (RBFN) training method, 312, 322
sensitivity analysis, 315
stopping criteria, 312
training log, 315
neural nets
  exporting, 467, 467, 468, 468, 469
neural network models
  generating filter nodes, 392
neural networks, 42, 311, 324, 388, 389, 444, 445, 445,
  confidence, 388
new features, 1
node, 848
node reference, 697
nodes, 21, 697
  adding, 61, 64
  adding to projects, 578, 579
  annotating, 70
  bypassing in a stream, 63
  connecting in a stream, 61
  deleting, 61
  deleting connections, 66
  duplicating, 66
  editing, 66
  introduction, 59
  loading, 88
  names, 699, 713, 714
  positioning, 706
  saving, 86
  scripting, 699, 704, 707, 709, 710, 713, 713,
  714, 717, 720
  setting options, 66
NODE specification, 609, 616
  HASH_DATA keyword, 800, 803
  ICON subsection, 803
  NAME keyword, 800, 801
  PALETTE keyword, 800, 802
  TITLE keyword, 800, 801
  TYPE keyword, 800
  TYPE keyword, 801
noisy data, 51
nominal regression, 348
normalized chi-square (apriori evaluation measure), 345
normalized data, 848
notifications
  setting options, 33
nulls, 155, 218, 223
null values
  quality node, 523
number function, 696
number parameters, 806, 815
numbers, 219, 664, 664
numeric functions, 674
objects
  properties, 585
ODBC, 106, 848
  bulk loading via ODBC, 536
ODBC output node. See database output node, 533
oneof, 678
opening
  models, 88
  nodes, 88
  output, 88
  output objects, 488
  projects, 579
  states, 88
  streams, 88
operator precedence, 668
optimization, 1
  options, 37
options, 31
  AnswerTree, 551
  display, 36
Excel, 551
for Clementine, 31
optimization, 37
SPSS, 550
stream properties, 73, 76, 78, 80
user, 33
OPTIONS subsection, 610, 619, 812
Oracle, 106
order
of input data, 141
ordering data, 134, 202
order merging, 135
order of execution
specifying, 572
outer join, 135
outlier, 848
output
scripting, 726
working with, 488
OUTPUT_DATA subsection, 614, 621, 826
OUTPUT_FIELDS subsection, 613, 620, 821
output browsers
menus, 489
output field, 848
output formats, 491
output manager, 488
output nodes, 487, 488, 493, 493, 494, 498, 504,
510, 518, 522, 527, 529, 531, 533, 539, 540,
544, 545, 546
output tab, 491
publish to web, 490
scripting properties, 781
output palette, 803
overfitting, 848
overlays, 246
data audit browser, 515
for graphs, 232
overwriting database tables, 534
PALETTE keyword, 802
palette manager
CEMI, 624
palettes, 848
overview, 24
panel, 232
parameter estimates
logistic regression, 397
parameters, 602, 669, 697, 701, 701, 702, 704, 713,
714, 738, 738, 740, 794, 796, 848
for streams, 80, 82
node properties, 569
scripting, 697, 699
setting for supernodes, 566, 566, 568
slot parameters, 569
PARAMETERS subsection, 610, 617, 805
field parameters, 807
flag parameters, 806
id parameters, 810
number parameters, 806
pathname parameters, 809
set parameters, 808
tempfile parameters, 809
text parameters, 806
part and partial correlations
linear regression, 338, 401
partial joins, 135, 139
passing samples, 128
passwords
database connections, 108
encoded, 732, 746
paste, 26
pathname parameters, 809, 815
PCA, 359, 848
percentages, 128
performance evaluation statistic, 504
period, 73
persistence
neural net node, 317, 319, 321, 322
pi, 676
plot node, 231
creating, 243, 246
using the graph, 251
plotting associations, 276, 278
PMML, 464, 465, 833
PMML models
exporting, 382
importing, 382, 384
point plots, 231, 243, 256
positioning nodes, 717
PowerPoint files, 578
precedence, 668
prediction, 848
PredictiveMarketing, 10, 481
predictor, 848
principal components analysis (PCA), 359, 403, 404, 406
printing
output, 489
streams, 66
prior probabilities
C&R tree node, 372
probabilities
in logistic regression, 393
probability, 848
process nodes, 607, 802, 802, 805
profit charts, 290, 299
projects, 575
adding objects, 579
annotating, 583
building, 579
classes view, 578
closing, 585
creating new projects, 579
CRISP-DM view, 576
folder properties, 583
generating reports, 586
object properties, 585
setting a default folder, 577
setting properties, 581
project tool, 848
promax rotation
factor/PCA node, 363
properties
common scripting, 741
filter nodes, 738
for data streams, 73
for scripting, 737
node, 569
project folder, 583
report phases, 587
scripting, 700, 738, 738, 740, 741, 780, 794
stream, 796
pruning, 848
pseudo R-square
logistic regression, 397
publisher node, 531
publish to web, 490
URL setting, 551
purple nodes, 37
python
bulk loading scripts, 536
quality browser
edit menu, 489
file menu, 489
generate menu, 524
generating filter nodes, 525
generating select nodes, 525
interpreting results, 524
quality node, 522
invalid value criteria, 523
output tab, 491
quality tab, 523
quantile, 848
quartile, 848
quartimax rotation
factor/PCA node, 363
query, 106, 106, 848
query editor, 110
QUEST, 384
quintile, 848
quotes
for database export, 534
radial basis function nets
exporting, 470, 471, 471, 472, 472
radial basis function network (RBFN), 312, 322, 850
random functions
oneof, 678
random0, 678
random samples, 127
range
statistics output, 520
range field type, 157
ranges, 150, 150, 151, 151
rank cases, 194
real functions, 696
real ranges, 157
reals, 663, 664, 850
reclassify node, 184, 187
generating from a distribution, 261
overview, 184, 188
Index

recode, 184, 184, 188
record counts, 132
record length, 101
record operations nodes, 125
aggregate node, 131
append node, 143
balance node, 129
distinct node, 145
merge node, 135
sample node, 127
scripting properties, 747
select node, 126
sort node, 134
record ops palette, 802
records, 41, 850
merging, 135
missing values, 226
refined models, 837, 850
externally executable, 838
internally executable, 838
refresh
source nodes, 73
regression, 334, 398
regression equation node, 398
regression trees, 367, 850
relational database, 850
rename statements, 717
renaming
output objects, 488
streams, 83
replacing field values, 181
report browser, 529
edit menu, 489
file menu, 489
report manager, 850
report node, 527
HTML, 492
output tab, 491
template tab, 527
reports
adding to projects, 578
generating, 586
setting properties, 587
resizing, 27
response charts, 290, 299
RESULTS subsection, 615, 833
retail analysis, 647
RETURN_CODE subsection, 615, 621, 832
ROI charts, 290, 299
rotation
of factors/components, 363
rough diamond, 850
round, 674
row, 850
row-wise binding, 536
R-squared change
linear regression, 338, 401
rule induction, 43, 329, 339, 341, 367, 850
rules
exporting, 466, 466, 466
ruleset node, 411, 412, 414, 415, 417, 426
rulesets, 850
evaluating, 73
rule supernode
generating from sequence rules, 460
sample node
overview, 127
setting options, 128
sampling, 56, 127, 850
sampling percentages, 128
SAS
import node, 113
setting import options, 114
transport files, 113
types of import files, 113
SAS data, 544
SAS export node, 544, 544
sav files, 111
saving
generated models, 382
multiple objects, 87
nodes, 86
output, 489
output objects, 488
states, 86, 87
streams, 86
scale factors, 130
scatterplots, 231, 243, 256, 850
scenario, 481, 484
schema
database output node, 535
Schwarz Bayesian Criterion
linear regression, 338
scoring, 850
evaluation chart options, 297
scoring data, 385
scripting, 39, 205, 714, 727, 850
abbreviations used, 740
batch mode, 606
checking, 601
CLEM expressions, 704
closing streams, 710
common properties, 741
creating nodes, 704
creating streams, 704
current object, 709, 710
deleting nodes, 710
dependencies, 598, 601, 603, 605
executing, 729
exit commands, 727
field operations nodes, 751
file output, 726
generated models, 722
get object, 709
graph nodes, 758
in batch mode, 594
interrupting, 729
language reference, 697
manipulating results, 725
modeling nodes, 767
models, 722, 723, 723
node commands, 717
node properties, 721
nodes, 713, 713
opening nodes, 707
operators, 699
output nodes, 781
overview, 597, 697
parameters, 701
record operations nodes, 747
saving streams, 707
setting parameters, 700
setting properties, 700
source nodes, 742
standalone scripts, 598
streams, 598, 711
supernodes, 572, 602, 724
supernode scripts, 598
syntax, 697, 698, 699, 699
user interface, 599, 599, 602, 604
scrolling
setting options, 76
sd2 (SAS) files, 113
searching
table browser, 496
security
encoded passwords, 732, 746
seed
random seed, 128
segmentation, 850
select, 850
selecting rows (cases), 126
selection criteria
linear regression, 338, 401
select node
generating, 251, 261, 266, 274, 284
overview, 126
self-organizing maps, 44, 324
sensitivity analysis, 850
neural net node, 315
neural networks, 389
sequence browser, 459
sequence detection, 45, 373
sequence functions, 688
sequence node, 373, 377
collection field(s), 374
data formats, 374
expert options, 377
field options, 374
generated sequence rules, 454, 455
ID field, 374
options, 376
predictions, 455
tabular versus transactional data, 377
time field, 374
sequence rules node
generating a rule supernode, 460
sequences, 850
  generated sequence rules, 456, 459
  sequence browser, 459
  sorting, 459
server
  command line arguments, 732
  default directory, 32
server mode, 22
session
  parameters, 82
set field type, 850
set globals node, 529
  settings tab, 530
set parameters, 808, 815
sets, 73
  transforming, 184, 187
set to flag node, 197, 198
set type, 150, 151, 158
SHEET subsection, 816
shortcut keys, 28
sign, 674
significance (statistical), 850
sin, 676
sinh, 676
skipchar, 679
skipchar_back, 679
slb files, 573
slot parameters, 569, 602, 699, 700, 713, 721, 737, 741, 850
solution publisher node, 531, 532
solutions template library, 89
sorting
  fields, 201
  records, 134
sort node
  overview, 134
source nodes, 607, 802, 802, 805
  database node, 106
  data mapping, 90
  fixed file node, 101
  instantiating types, 121
  overview, 97
  refreshing, 73
  SAS import node, 113
scripting properties, 742
  SPSS file node, 111
  user input node, 114, 116
  variable file node, 98
sources palette, 802
SPECFILE, 799
  CORE specification, 804
  MODEL specification, 836
  NODE specification, 800
special functions, 696
special variables, 709, 710
specification file, 607, 609
  backslashes, 799
  comments, 799
  notation, 800
SPSS
  launching from Clementine, 541, 546, 550
  valid field names, 542
SPSS export node, 540
  export tab, 541
SPSS file node
  overview, 111
SPSS output browser, 549
SPSS procedure node, 546, 547
SPSS Web Deployment Framework (SWDF)
  publishing output, 490
SQL, 106, 106, 110, 850
  generating from rulesets, 417
SQL generation, 37
sqrt, 674
ssd (SAS) files, 113
standalone scripts, 598, 599, 604
standard deviation, 850
  binning node, 195
  set globals node, 530
  statistics output, 520
standard deviation for aggregation, 132
standard error of mean
  statistics output, 520
states
  loading, 88
  saving, 86, 87
statistical models, 47
statistics, 850
  data audit node, 510
  matrix node, 498
statistics browser
  edit menu, 489
  file menu, 489
  generate menu, 520
  generating filter nodes, 522
  interpreting, 520
statistics node, 518
  correlation labels, 519
  correlations, 518
  output tab, 491
  settings tab, 518
  statistics, 518
status window, 24
stop execution, 26
stopping options
  C&R tree node, 371
storage, 104, 154, 155
  converting, 181, 183
stream canvas
  overview, 24
  settings, 76
streams, 21, 850
  adding nodes, 61, 64
  adding to projects, 578, 579
  annotating, 83
  building, 57, 58
  bypassing nodes, 63
  connecting nodes, 61
  execution, 85
  loading, 88
  multiset command, 737
  options, 73
  properties, 713, 796
  saving, 86
  scheduling, 591
  scripting, 597, 598, 599, 601, 704, 707, 709, 710, 711, 711, 712
  scripting properties, 711
  setting parameters, 80
strfloat, 696
string functions, 679, 696
strings, 217, 663, 665, 850
  scripting, 699
strinteger, 696
stripchar, 696
strmember, 679
strnumber, 696
structured properties, 738
subscr, 679
substringsubstring_between, 679
sum
  set globals node, 530
  statistics output, 520
summary data, 131
summary statistics
  data audit node, 510
summed values, 132
supernodes, 553, 598, 729, 737, 794
  creating, 557
  creating caches for, 570
  editing, 564
  loading, 573
  nesting, 560
  process supernodes, 555
  saving, 573
  scripting, 572, 599, 603, 697, 724
  setting parameters, 566, 566, 568
  source supernodes, 554
  terminal supernodes, 556
  types of, 554
  zooming in, 564
supervised learning, 850
support, 850
  apriori node, 342
  for sequences, 456
  GRI node, 340
  sequence node, 376
surrogates
  decision tree node, 422
Sybase, 106
symbolic field, 850
syntax
  for scripting, 697, 698, 699, 699
syntax tab
  SPSS procedure node, 547
synthetic data, 114
system
  command line arguments, 734
  options, 31
  requirements, 18
system-missing values, 223
Index

table
  reading from a database, 109
table browser
    edit menu, 489
    file menu, 489
    generate menu, 496
    reordering columns, 496
    searching, 496
    selecting cells, 496
table node, 493
    column justification, 494
    column width, 494
    format tab, 494
    output settings, 493
    settings tab, 493
table owner, 109

tables
  adding to projects, 578
  joining, 135
TAB subsection, 818

tabular data
  apriori node, 310
  sequence node, 374
tags, 135, 141
tan, 676
tanh, 676
target, 853
temp directory, 23
tempfile parameters, 809
template fields, 94
templates, 89, 90, 481, 484
  report node, 527
terminal nodes, 608, 802, 803, 805
testbit, 677
test metric
  neural net node, 315
text data, 98, 101
text files, 98
text parameters, 806, 815
thresholds
  viewing bin thresholds, 196
throughput reporting, 37
ties, 192
tiles, 192
time
  setting formats, 73
time_before, 673
time and date functions, 666, 667
time field
  sequence node, 374
time formats, 666, 667
time functions, 666, 667
  time_before, 684
  time_hours_difference, 684
  time_in_hours, 684
  time_in_mins, 684
  time_in_secs, 684
  time_mins_difference, 684
  time_secs_difference, 684
times, 217
time series, 199, 853
timestamp, 150
to_ functions
  to_date, 672
  to_integer, 672
  to_real, 672
  to_string, 672
  to_time, 672
  to_timestamp, 672
toolbar, 26
tpt (SAS) files, 113
training data sets, 127
train metric
  neural net node, 315
train net node
  See neural net node, 311
transactional data
  apriori node, 310
  sequence node, 374
transformations, 853
  reclassify, 184, 188
  recode, 184, 188
transparency, 232
tree map
  decision tree node, 423
tree viewer, 423
trigonometric functions, 676
true values, 159
truncating field names, 165, 167
TwoStep clustering, 48, 451, 452, 853
TwoStep cluster node, 364, 366, 450
   number of clusters, 366
   options, 366
   outlier handling, 366
   standardization of fields, 366
   type, 104, 853
   type attributes, 163
   TYPE keyword, 801
   type node, 224
      blank handling, 155
      clearing values, 119
      copying types, 163
      flag field type, 159
      instantiating, 153
      missing values, 227
      overview, 148
      range field type, 157
      set field type, 158
      setting direction, 162
      setting options, 150, 151
   unbiased data, 129
   undef, 228, 695
   undefined values, 137
   undo, 26
   uninstalling
      CEMI nodes, 626, 627
      Clementine, 21
   unique records, 145
   unmapping fields, 89
   unrefined models, 45, 837, 853
   unrefined rule node, 407, 408, 409, 411
   unsupervised learning, 48, 853
   unsupervised models, 324
   uppertolower, 679
   upstream, 853
   usage type, 104
   user input node, 853
      overview, 114
      setting options, 116
   user-missing values, 223
   user options, 33
   value labels, 111
   values, 206
      adding to CLEM expressions, 214
      reading, 154
   specifying, 155
   viewing from a data audit, 214
   variable file node, 98, 853
      setting options, 99
   variable labels, 111
   SPSS export node, 540
   variable names
      data export, 533, 539, 541, 544
   variables, 41, 704, 853
      scripting, 709, 710
   variance, 853
      statistics output, 520
   varimax rotation
      factor/PCA node, 363
   viewer tab
      decision tree node, 423
   views, 109
   vingtile, 853
   visual programming, 24, 853
   warnings, 78
      setting options, 33
   web node, 853
      adjusting thresholds, 287, 289
      appearance options, 282
      creating, 280
      overview, 276, 278
      using the graph, 284
   weighted least squares, 309
   weight fields, 309
   what's new, 1
      algorithms information, 7
      binning node, 3
      bulk loading, 12
      CEMI changes, 13
      cluster viewer, 7
      data audit node, 3
      deployment options, 10
      encoded password generator, 13
      evaluation chart improvements, 9
      field reorder node, 6
      in-database scoring, 12
      lift statistic, 9
      merge node enhancements, 7
      modeling enhancements, 7
      node properties, 13, 14
Index

reclassify node, 5
scripting, 13
white space
  quality node, 523
wizard
  accessing, 481, 484
  overview, 481, 484

XML
  exporting, 722
  x-y grid coordinates, 706
  zooming, 26, 564