

> Clementine® 10.0 User's Guide



For more information about SPSS® software products, please visit our Web site at <http://www.spss.com> or contact

SPSS Inc.

233 South Wacker Drive, 11th Floor

Chicago, IL 60606-6412

Tel: (312) 651-3000

Fax: (312) 651-3668

SPSS is a registered trademark and the other product names are the trademarks of SPSS Inc. for its proprietary computer software. No material describing such software may be produced or distributed without the written permission of the owners of the trademark and license rights in the software and the copyrights in the published materials.

The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of The Rights in Technical Data and Computer Software clause at 52.227-7013. Contractor/manufacture is SPSS Inc., 233 South Wacker Drive, 11th Floor, Chicago, IL 60606-6412.

General notice: Other product names mentioned herein are used for identification purposes only and may be trademarks of their respective companies.

Graphs powered by SPSS Inc.'s nViZn™ advanced visualization technology (<http://www.spss.com/sm/nvizn>).

This product contains software developed by the Apache Software Foundation. Copyright © 1999–2000 by the Apache Software Foundation. All rights reserved.

This product includes software developed by Eric Young (ey@mincom.oz.au). Copyright © 1995–1997 by Eric Young. All rights reserved.

This product includes the International Components for Unicode. Copyright © 1995–2002 by IBM Corporation and others. All rights reserved.

This product includes the Java Access Bridge. Copyright © by Sun Microsystems Inc. All rights reserved. See the License for the specific language governing permissions and limitations under the License.

Microsoft and Windows are registered trademarks of Microsoft Corporation.

IBM, DB2, and Intelligent Miner are trademarks of IBM Corporation in the U.S.A. and/or other countries.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates.

UNIX is a registered trademark of The Open Group.

DataDirect and SequeLink are registered trademarks of DataDirect Technologies.

This product includes code licensed from RSA Security, Inc. Some portions licensed from IBM Corporation are available at <http://oss.software.ibm.com/icu4j/>.

C5.0 Copyright © 1997–2004 by RuleQuest Research Pty Ltd. All rights reserved.

Apriori—Finding Association Rules/Hyperedges with the Apriori Algorithm. Copyright © 1996–2003 by Christian Borgelt (<http://fuzzy.cs.uni-magdeburg.de/~borgelt/doc/apriori/apriori.html>).

Portions of the software are developed and owned by:

Copyright © 1999, 2000 by Boris Fomitchev.

This material is provided “as is,” with absolutely no warranty expressed or implied. Any use is at your own risk. Permission to use or copy this software for any purpose is hereby granted without fee, provided the above notices are retained on all copies. Permission to modify the code and to distribute modified code is granted, provided the above notices are retained, and a notice that the code was modified is included with the above copyright notice.

Copyright © 1994 by Hewlett-Packard Company.

Copyright © 1996, 1997 by Silicon Graphics Computer Systems, Inc.

Copyright © 1997 by Moscow Center for SPARC Technology.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Hewlett-Packard Company makes no representations about the suitability of this software for any purpose. It is provided “as is” without express or implied warranty.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Silicon Graphics makes no representations about the suitability of this software for any purpose. It is provided “as is” without express or implied warranty.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Moscow Center for SPARC Technology makes no representations about the suitability of this software for any purpose. It is provided “as is” without express or implied warranty.

Project phases are based on the CRISP-DM process model. Copyright © 1997–2003 by CRISP-DM Consortium (<http://www.crisp-dm.org>).

Copyright © 1995–2001 by Corporation for National Research Initiatives. All rights reserved.

Copyright © 1991–1995 by Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Copyright © 1999–2004 by Sourceforge JACOB Project. All rights reserved. Originator: Dan Adler (<http://danadler.com>). The JACOB Project (including full source code) can be found at <http://sourceforge.net/projects/jacob-project/>.

Copyright © 2001–2005 by JGoodies. Founder: Karsten Lentzsch. All rights reserved.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Jgoodies or Karsten Lentzsch nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Castor Copyright © 1999–2004 by Intalio Inc. and others. All rights reserved. Redistribution and use of this software and associated documentation (“Software”), with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain copyright statements and notices. Redistributions must also contain a copy of this document.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name “ExoLab” must not be used to endorse or promote products derived from this Software without prior written permission of Intalio Inc. For written permission, please contact info@exolab.org.
4. Products derived from this Software may not be called “Castor” nor may “Castor” appear in their names without prior written permission of Intalio Inc. Exolab, Castor, and Intalio are trademarks of Intalio Inc.
5. Due credit should be given to the ExoLab Project (<http://www.exolab.org/>).

THIS SOFTWARE IS PROVIDED BY INTALIO AND CONTRIBUTORS “AS IS” AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL INTALIO OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 2000 by *BeOpen.com*. All rights reserved.

Language Weaver Statistical Machine Translation Software (SMTS). Copyright © 2004, 2005, Language Weaver, Inc. All rights reserved.

libTextCat. Terms of use covered under the BSD License. Copyright © 1998 by the Regents of the University of California. All rights reserved.

Copyright © 1990, 1993, 1994 by the Regents of the University of California. All rights reserved.

ICU 1.8.1 and later. Copyright © 1995–2003 by International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. THE SOFTWARE IS PROVIDED “AS IS,” WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

Clementine® 10.0 User’s Guide

Copyright © 2005 by Integral Solutions Limited.

All rights reserved.

Printed in the United States of America.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means—electronic, mechanical, photocopying, recording, or otherwise—without the prior written permission of the publisher.

Preface

Clementine is the SPSS enterprise-strength data mining workbench. Clementine helps organizations to 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 to apply their 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.

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	<i>About Clementine</i>	1
	Clementine Client/Server	1
2	<i>New Features</i>	3
	New Features in Clementine 10.	3
	Clementine Documentation.	7
3	<i>Clementine Overview</i>	9
	Getting Started	9
	Starting Clementine	9
	Clementine at a Glance.	12
	Clementine Interface	13
	Using the Mouse in Clementine	17
	Using Shortcut Keys	17
	Getting Help in Clementine.	19
	Setting Clementine Options.	20
	System Options	20
	Setting Default Directories.	22
	Setting User Options	22
	Printing.	31
	Automating Clementine.	32

4 *Understanding Data Mining*

33

Data Mining Overview	33
Screening Models	35
Decision Trees (Rule Induction)	38
Neural Networks	42
Kohonen Networks	43
Statistical Models.	44
Clustering Models.	46
Association Rules.	47
Text Mining.	50
Assessing Potential Data Mining Applications	50
Is the Data Available?	50
Does the Data Cover the Relevant Attributes?	51
Is the Data Noisy?	51
Is There Enough Data?.	51
Is Expertise on the Data Available?	52
A Strategy for Data Mining	52
The CRISP-DM Process Model	53
Tips	55
Induction, Neural Net, or Statistical Models?.	55
Is the Data Balanced?	56
Sampling	56
Screening Fields for Analysis.	56
Collecting Exceptions	57
Data Mining Examples	57
Using Clementine Application Templates	57

61

6 Handling Missing Values 103

7 Building CLEM Expressions 111

ix

Working with Numbers	119
Working with Times and Dates	120
Summarizing Multiple Fields	121
Using the Expression Builder	122
Accessing the Expression Builder	123
Creating Expressions	124
Selecting Functions	124
Selecting Fields, Parameters, and Global Variables	125
Viewing or Selecting Values.	126
Checking CLEM Expressions	127

8 CLEM Language Reference

129

CLEM Reference Overview	129
CLEM Datatypes	129
Integers.	130
Reals	130
Characters	131
Strings.	131
Lists.	131
Fields.	132
Dates.	132
Time	133
Operator Precedence	134
Functions Reference.	135
Conventions in Function Descriptions	136
Information Functions	137
Conversion Functions	138
Comparison Functions	139
Logical Functions.	140
Numeric Functions	141
Trigonometric Functions	142

Probability Functions	143
Bitwise Integer Operations	143
Random Functions	145
String Functions.	145
SoundEx Functions	150
Date and Time Functions	151
Sequence Functions	155
Global Functions	161
Functions Handling Blanks and Null Values	162
Special Fields	163

9 *Projects and Reports* 165

Introduction to Projects	165
CRISP-DM View.	166
Classes View.	168
Building a Project.	169
Creating a New Project	169
Adding to a Project	169
Transferring Projects to Predictive Enterprise Repository.	171
Setting Project Properties	172
Annotating a Project	174
Object Properties.	176
Closing a Project	176
Generating a Report	177
Saving and Exporting Generated Reports.	180

10 Deploying Models and Streams

185

Predictive Applications Wizard	186
Before Using the Predictive Applications Wizard	186
Converting Scores to Propensities before Exporting.	188
Step 1: Predictive Applications Wizard Overview	189
Step 2: Selecting a Terminal Node	190
Step 3: Selecting a UCV Node	191
Step 4: Specifying a Package.	193
Step 5: Generating the Package.	194
Step 6: Summary	196
Exporting to Cleo	196
Cleo Wizard Overview	196
Cleo Stream Prerequisites	197
Step 1: Cleo Wizard Overview Screen	198
Exporting Models as PMML	199
Importing Models Saved as PMML	200
Model Types Supporting PMML	202

11 Predictive Enterprise Repository

203

Connecting to Predictive Enterprise Repository.	204
Storing Objects.	206
Retrieving Stored Objects	208
Browsing Repository Content	210
Object Properties.	212
Deleting Repository Objects.	215
Searching Repository Content	216
Adding and Removing Folders	219

Application Examples

12 Application Examples 223

13 Screening Predictors (Feature Selection) 225

Building the Stream	226
Building the Models	228
Comparing the Results	229
Summary	231

14 Fraud Screening (Anomaly Detection/Neural Net) 233

Preliminary Screening	234
Data Investigation.	241
Training a Neural Network	246
Anomaly Detection Revisited	250
Summary	252

15 Market Basket Analysis (Rule Induction/C5.0) 255

Accessing the Data	255
Discovering Affinities in Basket Contents	257
Profiling the Customer Groups	260
Summary	263

16 Retail Sales Promotion (Neural Net/C&RT) 265

Examining the Data	265
Learning and Testing	269

17 Condition Monitoring (Neural Net/C5.0) 271

Examining the Data	272
Data Preparation	275
Learning	276
Testing	277

Appendices

A Accessibility in Clementine 279

Overview of Clementine Accessibility	279
Types of Accessibility Support	279
Accessibility for the Visually Impaired	279
Accessibility for Blind Users	281
Keyboard Accessibility	282
Using a Screen Reader	288
Accessibility in the Tree Builder	289
Tips for Use	289
Interference with Other Software	290
JAWS and Java	291
Using Graphs in Clementine	291

<i>B</i>	<i>Unicode Support</i>	<i>293</i>
	Unicode Support in Clementine	293
<i>C</i>	<i>Features Added in Previous Releases</i>	<i>295</i>
	New Features in Clementine 9.0	295
	New Features in Release 8.5.	298
	New Features in Release 8.0.	299
	<i>Glossary</i>	<i>301</i>
	<i>Index</i>	<i>315</i>

About Clementine

Clementine is a data mining workbench that enables you to quickly develop predictive models using business expertise and deploy them into business operations to improve decision making. Designed around the industry-standard CRISP-DM model, Clementine supports the entire data mining process, from data to better business results.

Clementine Client/Server

The Clementine Client/Server release distributes client requests for resource-intensive operations to powerful server software, resulting in faster performance on larger data sets. Additional products or updates beyond those listed here may also be available. For the most current information, see the Clementine Web page (<http://www.spss.com/clementine/>).

Clementine Client. Clementine Client is a functionally complete version of the product that is installed and run on the user's desktop computer. It can be run in local mode as a stand-alone product or in distributed mode along with Clementine Server for improved performance on large data sets.

Clementine Server. The Clementine Server runs continually in distributed analysis mode together with one or more Clementine Client installations. This provides superior performance on large data sets, because memory-intensive operations can be done on the server without downloading data to the client computer. At least one client installation must be present to run an analysis.

Clementine Solution Publisher. Clementine Solution Publisher is an add-on component that enables organizations to publish Clementine streams for use outside of the standard Clementine environment. Published streams can be executed by using the Clementine Solution Publisher Runtime, which can be distributed and deployed as needed. Solution Publisher is installed along with Clementine Client but requires a separate license to enable the functionality.

Text Mining for Clementine. Text Mining for Clementine is an add-on product that uses linguistic methods to extract key concepts from text, based on context. This information can be combined with existing structured data, such as demographics, and applied to modeling using Clementine's full suite of data mining tools to yield better and more focused decisions. The new release of Text Mining for Clementine is fully integrated with Clementine and can be deployed with applications such as PredictiveCallCenter to support real-time scoring of unstructured data. The new Concept Extraction Browser streamlines the modeling process, allowing you to select concepts from a visual display and merge those concepts directly into your data. You can also use Text Mining Builder with Text Mining for Clementine, which allows you to fine-tune the extraction process through an easy-to-use graphical environment. Custom dictionaries for specific domains, such as CRM and genomics, are also included. Text Mining for Clementine is installed along with Clementine Client but requires a separate license to enable the functionality.

Clementine Batch. Batch mode allows long-running or repetitive tasks to be performed without your intervention and without the presence of the user interface on the screen. For customers who want to run Clementine exclusively in this manner, a special version of Clementine Client that supports only batch mode is available. Clementine Batch provides the complete analytical capabilities of the standard Clementine Client but without access to the regular user interface. It can be run in local mode as a stand-alone product or in distributed mode along with Clementine Server.

Web Mining for Clementine. Web Mining for Clementine is an add-on module that allows analysts to perform *ad hoc* predictive Web analysis within Clementine's intuitive visual workflow interface. Powered by proven NetGenesis Web analytics technology, Web Mining for Clementine transforms raw Web data into analysis-ready business events that allow you to segment users, understand the pathways and affinities charted by users as they navigate your site, and predict user propensity to convert, buy, or churn.

New Features

New Features in Clementine 10.0

Clementine 10.0 delivers improved performance and scalability, new modeling nodes and statistics, expanded access to data, enhanced support for projects and reporting, Linux support, and a number of other improvements.

New Nodes in Clementine 10.0

The following nodes have been added in Clementine 10.0:



The Feature Selection node screens predictor fields for removal based on a set of criteria (such as the percentage of missing values); then it ranks the importance of remaining predictors relative to a specified target. For example, given a data set with hundreds of potential predictors, which are most likely to be useful in modeling patient outcomes?



The Anomaly Detection node identifies unusual cases, or outliers, that do not conform to patterns of “normal” data. With this node, it is possible to identify outliers even if they do not fit any previously known patterns and even if you are not sure exactly what you are looking for.



The Means node compares the means between independent groups or between pairs of related fields to test whether a significant difference exists. For example, you could compare mean revenues before and after running a promotion or compare revenues from customers who did not receive the promotion with those who did.



The Excel Import node imports data from any version of Microsoft Excel. An ODBC data source is not required.



The Dimensions Data Import node imports survey data based on the Dimensions Data Model used by SPSS market research products. The Dimensions Data Library must be installed to use this node.



The Restructure node converts a set or flag field into a group of fields that can be populated with the values of yet another field. For example, given a field named *payment type*, with values of *credit*, *cash*, and *debit*, three new fields would be created (*credit*, *cash*, *debit*), each of which might contain the value of the actual payment made.



The Time Intervals node specifies intervals and creates labels (if needed) for modeling time series data. If values are not evenly spaced, the node can pad or aggregate values as needed to generate a uniform interval between records.



The Time Plot node displays one or more sets of time series data. Typically, you would first use a Time Interval node to create a *TimeLabel* field, which would be used to label the x axis.



The Transpose node swaps the data in rows and columns so that records become fields and fields become records.

Performance and Scalability

Multi-threaded sorting and merging of records. Support for parallel processing to speed up sorting, merging, and binning operations, as well as the ability to specify a pre-sorted field in order to avoid sorting it again. In addition, enhancements to the merge node allow the user to specify which inputs should be cached and which are pre-sorted.

Database caching. For streams executed in the database, data can be cached midstream to a temporary table in the database rather than to the file system. When combined with SQL optimization, this may result in significant gains in performance. For example, the output from a stream that merges multiple tables to create a data mining view may be cached and reused as needed. With database caching enabled, simply right-click any non-terminal node to cache data at that point, and the cache is automatically created directly in the database the next time the stream is executed. This allows SQL to be generated for downstream nodes, further improving performance. (Alternatively, this option can be disabled if needed—for example, if policies or permissions preclude data being written to the database. If database caching or SQL optimization is not

enabled, the cache will be written to the file system instead.) For more information, see “Caching Options for Nodes” in Chapter 5 on p. 71.

Database indexing. You can now create indexes on database tables exported from Clementine to improve performance. You can specify the fields you want to include in each index and customize other options to meet the needs of specific databases.

Multi-threaded C5.0. Ability to take advantage of multiple processors during model building.

Analytical Enhancements

New modeling nodes. The Feature Selection node can be used to locate fields and records that are most likely to be of interest in modeling, and the Anomaly Detection node can be used to identify outliers that do not fit known patterns.

Means node. The Means node compares the means between independent groups, or between pairs of related fields, to test whether a significant difference exists.

Data Access and Preparation

Excel import/export. A new import node has been added. In addition, the Excel Export node has been extended to support more seamless data exchange between Clementine and Excel. For example, you can export the scores generated by a model to an Excel worksheet and automatically launch Excel when the stream is executed. The Publisher Node has also been extended to include support for exporting data to Excel.

Dimensions data import. You can now import survey data saved in the Dimensions format used by SPSS survey research products.

Restructure node. Offering a more flexible alternative to the Set to Flag node, the Restructure node can be used to generate multiple fields based on the values of a set or flag field.

Time series data. The Time Interval node specifies intervals and creates labels (if needed) for modeling time series data. If values are not evenly spaced, the node can pad or aggregate values as needed to generate a uniform interval between records. You can also transpose data using the Transpose node and plot your series using the Time Plot node.

New string functions. A number of features with an eye toward cleaning and sorting, including substring substitution, searching and matching, whitespace removal, string truncation, creation, and padding. For more information, see “String Functions” in Chapter 8 on p. 145.

Binning by equal sum of value. The Binning node now offers an option to automatically assign records to bins (categories) so that the sum of values in each bin (rather than the number of records) is roughly equal.

Projects and Reporting

Exporting projects to the Predictive Enterprise Repository. You can now store projects in addition to streams, models, and output objects, enhancing your ability to manage these objects within the larger predictive enterprise. For more information, see “Predictive Enterprise Repository” in Chapter 11 on p. 203.

Enhanced reporting. Graphs, tables, and other output objects can now be added to reports and exported to Microsoft Office applications or as HTML for publishing to the Web. For more information, see “Introduction to Projects” in Chapter 9 on p. 165.

Additional Enhancements

Linux support. Clementine 10.0 Server adds support for Red Hat Enterprise ES Linux ES 4 for 32-bit processor (x32).

LOESS smoother. For both two- and three-dimensional scatterplots, you can now choose to overlay a smoothed fit line computed using locally weighted iterative robust least squares regression (LOESS). This produces a series of “local” regression lines that are then joined to create a smooth curve.

Chi-square added to Matrix node. For a cross-tabulation of two categorical fields, the global Pearson chi-square is also displayed below the table.

CLEM functions for manipulating survey data. A number of functions that return summary statistics across multiple fields have been added. These functions may be particularly useful in analyzing survey data, where related responses to a question may be stored in multiple fields. For more information, see “Summarizing Multiple Fields” in Chapter 7 on p. 121.

Clementine Documentation

Complete documentation for Clementine Client is available on the product CD in both online Help (HTML) and PDF formats. The comprehensive online Help system and PDF documents can also be accessed from the Windows Start menu (Start > All Programs > Clementine 10.0 > Documentation).

- **Clementine 10.0 User's Guide.** General introduction to using Clementine, including overviews and examples.
- **Clementine 10.0 Node Reference.** Comprehensive descriptions of all nodes.
- **Clementine 10.0 Scripting, Automation, and CEMI Reference.** Information on automating and extending the system through scripting, batch mode execution, and the Clementine External Module Interface (CEMI).
- **Clementine 10.0 In-Database Mining Guide.** Information on how to leverage the power of your database to improve performance and extend the range of analytical capabilities through third-party algorithms.
- **Clementine 10.0 Server Administrator's Guide.** Information on how to configure and administer Clementine Server.
- **Clementine 10.0 Algorithms Guide.** Description of the mathematical foundations of the modeling methods used in Clementine.
- **Text Mining for Clementine 3.1 User's Guide.** Documentation for the Text Mining add-on, which provides the ability to extract structured data from unstructured information for further analysis. A separate license is required to use this product.
- **CRISP-DM 1.0 Guide.** Step-by-step guide to data mining using the CRISP-DM methodology.
- **SPSS Command Syntax Reference.** Documentation of SPSS commands (available through the SPSS Procedure node in Clementine).
- **Online Help and tutorials.** Comprehensive online Help is available throughout the system, along with introductory tutorials to help you get started. For more information, see "Getting Help in Clementine" in Chapter 3 on p. 19.

Additional documentation for Clementine Server, Clementine Batch, and Clementine Solution Publisher is distributed on the CD-ROM for each product.

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.

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 successfully installed Clementine, 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 or launch Clementine in batch mode. You can launch Clementine from both client and server computers by using the following steps:

- ▶ Open a DOS window or command-prompt window.
- ▶ Type the command `clementine` or `clemb` as well as any arguments (flags) used to load streams, execute scripts, connect to a server, or specify other parameters.

For more information, see the *Scripting, Automation, and CEMI Reference* (available under `\documentation\English_US\` on the Clementine Client CD-ROM, or from the Windows Start menu by choosing Start > [All] Programs > SPSS Clementine 10.0 > Documentation.)

Connecting to a Server

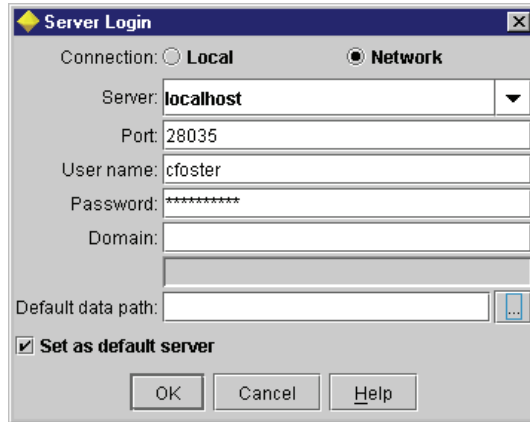
Clementine is a client-server application and can be run against the local computer or a server of your specification. The current connection status is displayed at the bottom left of the Clementine window.

To connect to a server:

- ▶ Double-click the connection status area of the Clementine window.
or
- ▶ From the Tools menu, choose Server Login.

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

Figure 3-1
Server Login dialog box

The image shows a 'Server Login' dialog box with a title bar containing a yellow diamond icon and a close button. The dialog has two radio buttons for 'Connection': 'Local' (unselected) and 'Network' (selected). Below this are several text input fields: 'Server' (containing 'localhost'), 'Port' (containing '28035'), 'User name' (containing 'cfooster'), 'Password' (containing '*****'), and 'Domain' (empty). There is also a 'Default data path' field with an ellipsis button to its right. At the bottom left is a checked checkbox labeled 'Set as default server'. At the bottom right are three buttons: 'OK', 'Cancel', and 'Help'.

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 Server 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.

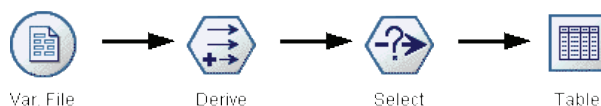
All temp files will now be written to this new directory.

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 3-2
A simple stream

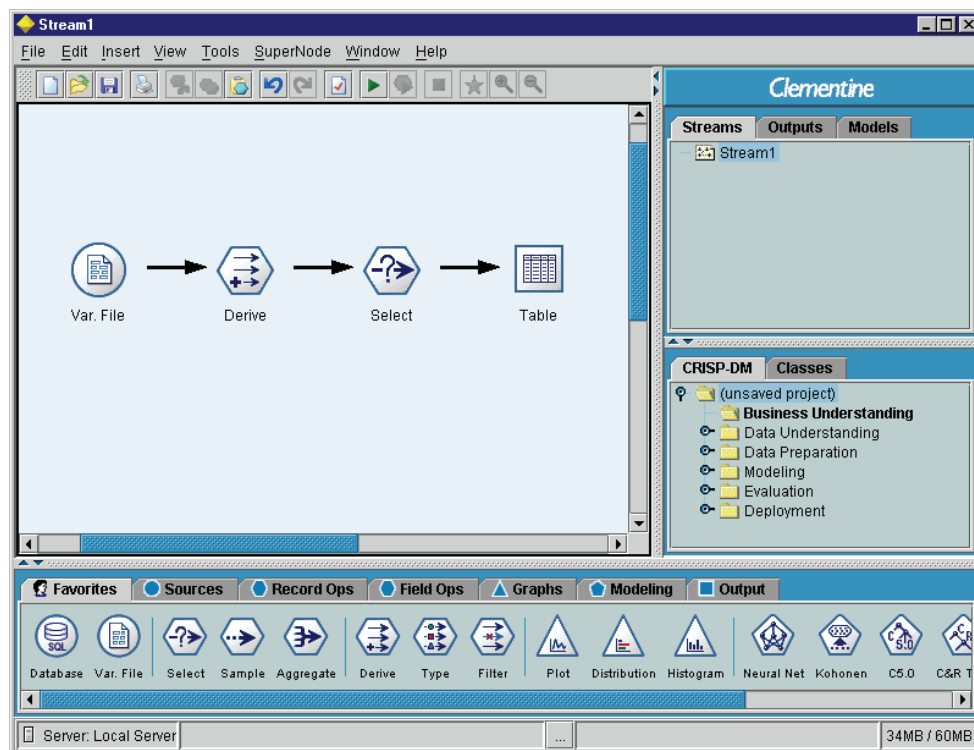


At each point in the data mining process, Clementine's visual interface invites your specific business expertise. Modeling algorithms, such as prediction, classification, segmentation, and association detection, ensure powerful and accurate models. Model results can easily be deployed and read into databases, 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.

Figure 3-3
Clementine user interface



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 on 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 9 on p. 165.

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 of 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
	Open Clementine Application Templates (CATs)		Cut & move to clipboard
	Copy to clipboard		Paste selection
	Undo last action		Redo
	Edit stream properties		Execute current stream
	Execute stream selection		Stop stream (Activated only during stream execution)



Add SuperNode



Zoom in (SuperNodes only)

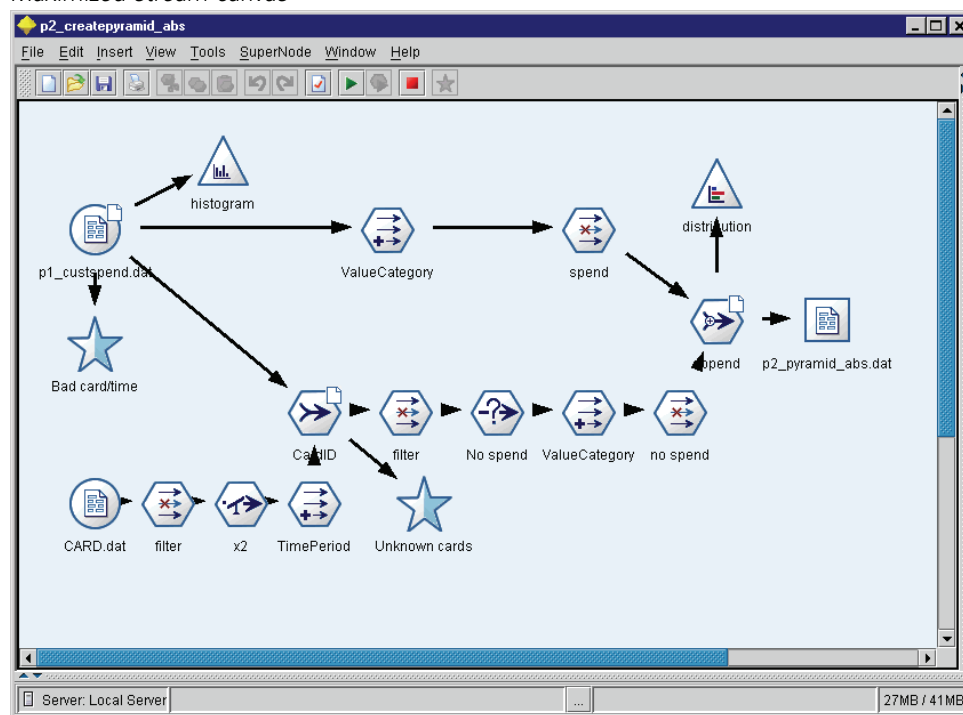


Zoom out (SuperNodes only)

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 3-4
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 3-1
Supported shortcut keys

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

Table 3-2
Supported shortcuts for old hot keys

Shortcut Key	Function
Ctrl-Alt-Z	Zoom out
Ctrl-Alt-D	Duplicate node
Ctrl-Alt-L	Load node
Ctrl-Alt-R	Rename node
Ctrl-Alt-U	Create User Input node
Ctrl-Alt-C	Toggle cache on/off
Ctrl-Alt-F	Flush cache
Ctrl-Alt-X	Expand SuperNode

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 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.
- **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.
- **PDF Manuals.** Complete documentation is available in the *\documentation* folder on each product CD and can also be accessed from the Windows Start menu by choosing Start > [All] Programs > SPSS Clementine 10.0 > Documentation. For more information, see "Clementine Documentation" in Chapter 2 on p. 7.

Microsoft Internet Explorer Settings

Most Help features in this application use technology based on Microsoft Internet Explorer. Some versions of Internet Explorer (including the version provided with Microsoft XP, Service Pack 2) will by default block what it considers “active content” in Internet Explorer windows on your local computer. This default setting may result in some blocked content in Help features. To see all Help content, you can change the default behavior of Internet Explorer.

- ▶ From the Internet Explorer menus choose:
Tools
Internet Options...
- ▶ Click the Advanced tab.
- ▶ Scroll down to the Security section.
- ▶ Select (check) Allow active content to run in files on My Computer.

Setting Clementine Options

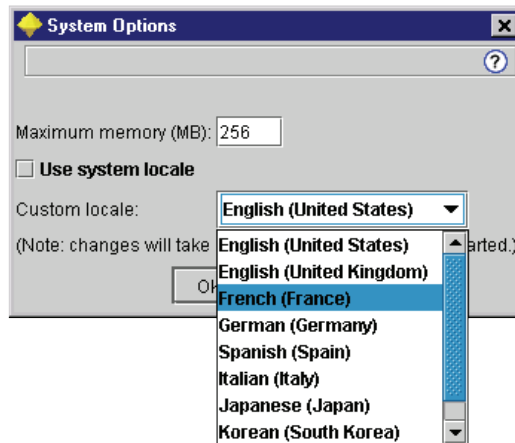
There are several ways to customize and set options for Clementine:

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

System Options

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

Figure 3-5
System Options dialog box



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 non-terminal 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 5 on p. 71.

- 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 5 on p. 77.
- 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.

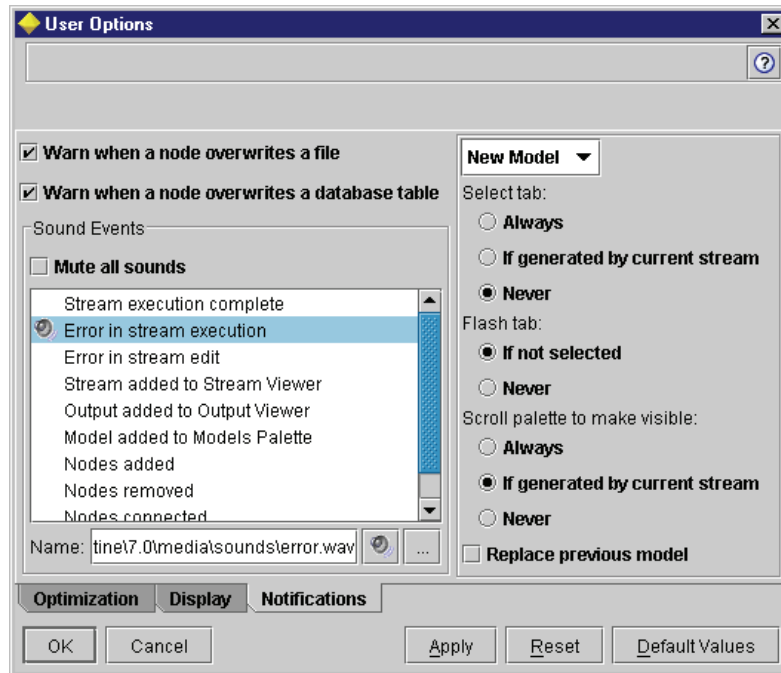
- PMML export options used when exporting models to Predictive Model Markup Language (PMML).
- User or author information, such as your name, initials, and e-mail address. This information may be displayed on the Annotations tab for nodes and for other objects that you create.

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.

Figure 3-6
Setting notification options



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.

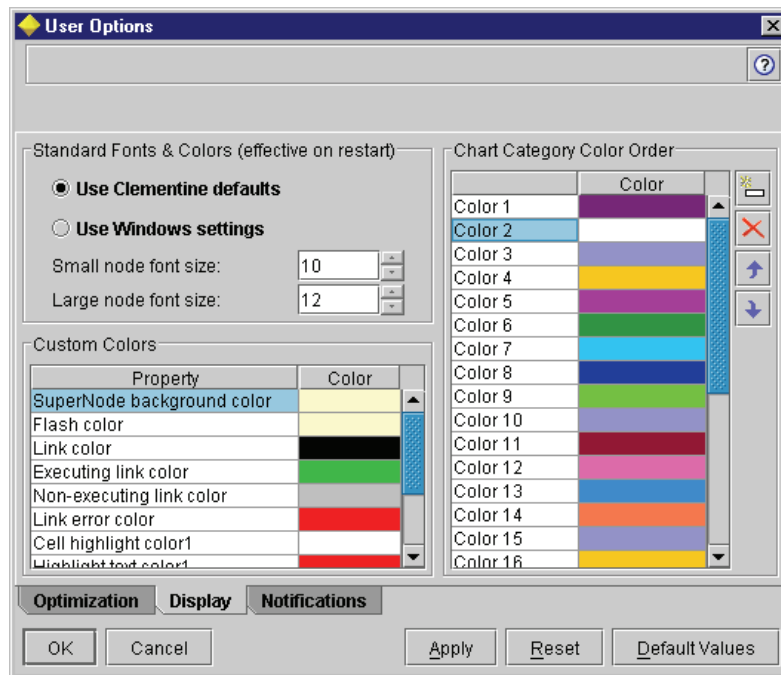
Replace previous model (New Model only). Select to overwrite previous iterations of the same model.

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 3-7
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 the 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.

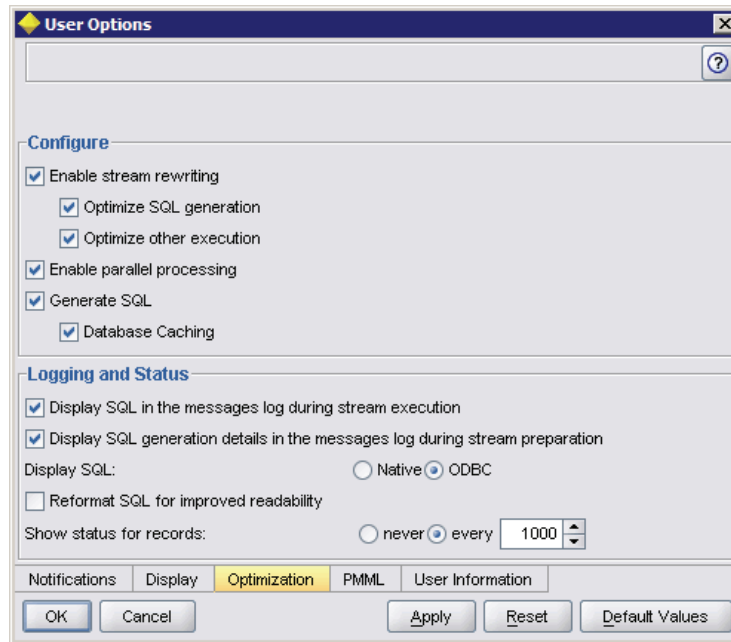
Chart Category Color Order. This table lists the currently selected colors used for display in newly created graphs. The order of the colors reflects the order in which they will be used in the chart. For example, if a set field used as a color overlay contains four unique values, then only the first 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

The Optimization tab in the User Options dialog box allows you to optimize Clementine performance during stream execution. Note that the Server optimization settings in *options.cfg* override any settings in the Client version.

Figure 3-8
Optimization settings



Enable stream rewriting. Select this option to enable stream rewriting in Clementine. Two types of rewriting are available, and you can select one or both. Stream rewriting reorders the nodes in a stream behind the scenes for more efficient execution by Clementine Server without altering stream semantics.

- **Optimize SQL generation.** This option allows Clementine to reorder nodes in the stream so that more operations can be pushed back using SQL generation for execution in the database. When it finds a node that cannot be rendered into SQL, the optimizer will look ahead to see if there are any downstream nodes that can be rendered into SQL and that can be safely moved in front of the problem node without affecting the stream semantics. Not only can the database perform operations more efficiently than Clementine but such pushbacks act to reduce the size of the data set that is returned to Clementine for processing. This, in turn, can reduce network traffic and speed stream operations.
- **Optimize other execution.** This method of stream rewriting increases the efficiency within Clementine of operations 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.

Enable parallel processing. When running Clementine on a computer with multiple processors, this option allows the system to balance the load across those processors, which may result in faster performance. Use of the following nodes may benefit from parallel processing: C5.0, Merge (by key), Sort, Bin (rank and tile methods), and Aggregate (using one or more key fields).

Database Caching. For streams executed in the database, data can be cached midstream to a temporary table in the database rather than to the file system. When combined with SQL optimization, this may result in significant gains in performance. For example, the output from a stream that merges multiple tables to create a data mining view may be cached and reused as needed. With database caching enabled, simply right-click any non-terminal node to cache data at that point, and the cache is automatically created directly in the database the next time the stream is executed. This allows SQL to be generated for downstream nodes, further improving performance. (Alternatively, this option can be disabled if needed—for example, if policies or permissions preclude data being written to the database. If database caching or SQL optimization is not enabled, the cache will be written to the file system instead.) For more information, see “Caching Options for Nodes” in Chapter 5 on p. 71.

Generate SQL. Select this option to enable SQL optimization, allowing stream operations to be pushed back to the database by using SQL code to generate execution processes, which may improve performance. To further improve performance, Optimize SQL generation can also be selected, allowing Clementine to maximize the number of operations pushed back to the database. When operations for a node have been passed back to the database, the node will be highlighted in purple during execution.

Note: Due to minor differences in SQL implementation, streams executed in a database may return slightly different results than when executed in Clementine. These differences may also vary depending on the database vendor, for similar reasons.

Logging and Status

Display SQL in the messages log during stream execution. Specifies whether SQL generated while executing the stream is passed to the message log.

Display SQL generation details in the messages log during stream preparation. During stream preview, specifies whether a preview of SQL that would be executed is passed to the messages log.

Display SQL. Specifies whether any SQL that is displayed in the log should contain native SQL functions or standard ODBC functions of the form {fn FUNC(...)} as generated by Clementine. The former relies on ODBC driver functionality that may not be implemented. For example, this control would have no effect for SQL Server.

Reformat SQL for improved readability. Specifies whether Clementine should reformat to improve the readability of any SQL displayed in the log.

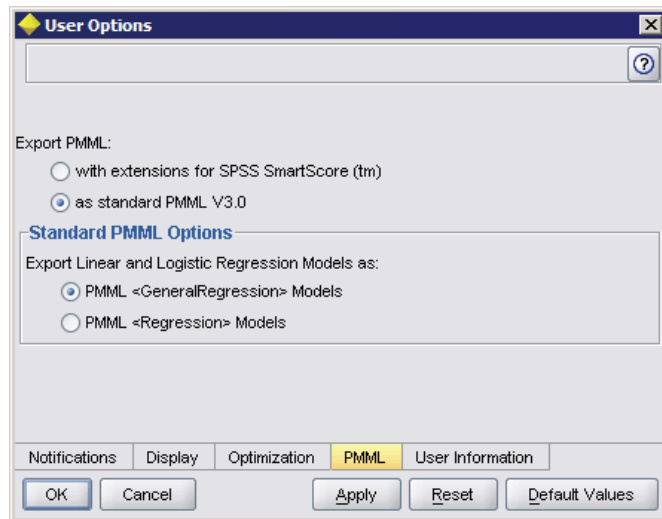
Show status for records. Selects whether Clementine should report records as they arrive at terminal nodes. Specify a number that is used for updating the status every *N* records.

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

Setting PMML Export Options

On the PMML tab, you can control how Clementine exports models to Predictive Model Markup Language (PMML).

Figure 3-9
Setting PMML export options



Export PMML. Here you can configure variations of PMML that work best with your target application.

- Select with extensions for SPSS SmartScore to allow PMML extensions for special cases where there is no standard PMML equivalent. Note that in most cases this will produce the same result as standard PMML.
- Select as standard PMML V3.0 to export PMML that adheres as closely as possible to the PMML standard.

Standard PMML Options. When standard PMML is selected above, you can choose one of two valid ways to export linear and logistic regression models:

- As PMML <GeneralRegression> models.
- As PMML <Regression> models.

Printing

The following objects can be printed in Clementine:

- Stream diagrams
- Graphs
- Tables
- Reports (from the Report node and Project Reports)
- Scripts (from the stream properties, Standalone Script, or SuperNode script dialog boxes)
- Models (Model browsers, dialog box tabs with current focus, tree viewers)
- Annotations (using the Annotations tab for output)

To print an object:

- To print without previewing, click the Print button on the toolbar.
- To set up the page before printing, select Page Setup from the File menu.
- To preview before printing, select Print Preview from the File menu.
- To view the standard print dialog box with options for selecting printers, and specifying appearance options, select Print from the File menu.

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 7 on p. 111.
- **Scripting** is a powerful tool for automating 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 the *Scripting, Automation, and CEM Reference* (available under `\documentation\English_US\` on the Clementine Client CD-ROM or from the Windows Start menu by choosing Start > [All] Programs > SPSS Clementine 10.0 > Documentation.)
- **Batch mode** enables you to use Clementine in a non-interactive 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 the *Scripting, Automation, and CEM Reference*.

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.

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 technique for building these models is called **machine learning**, or **modeling**.

Modeling Techniques

Clementine includes a number of machine-learning and modeling technologies, which can be roughly grouped according to the types of problems they are intended to solve: prediction, clustering, and association.

- Predictive modeling methods include decision trees, neural networks, and statistical models.
- Clustering models focus on identifying groups of similar records and labeling the records according to the group to which they belong. Clustering methods include Kohonen, K-Means, and TwoStep.

- Association rules associate a particular conclusion (such as the purchase of a particular product) with a set of conditions (the purchase of several other products).
- Screening models can be used to screen data to locate fields and records that are most likely to be of interest in modeling, and identify outliers that may not fit known patterns. Available methods include feature selection and anomaly detection.

Data Manipulation and Discovery

Clementine 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 an initial audit including graphs and statistics. 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.

Typical Applications

Typical applications of data mining techniques include the following:

Direct mail. Determine which demographic groups have the highest response rate. Use this information to maximize the response to future mailings.

Credit scoring. Use an individual's credit history to make credit decisions.

Human resources. Understand past hiring practices and create decision rules to streamline the hiring process.

Medical research. Create decision rules that suggest appropriate procedures based on medical evidence.

Market analysis. Determine which variables, such as geography, price, and customer characteristics, are associated with sales.

Quality control. Analyze data from product manufacturing and identify variables determining product defects.

Policy studies. Use survey data to formulate policy using decision rules to select the most important variables.

Health care. User surveys and clinical data can be combined to discover variables that contribute to health.

Terminology

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**.

Screening Models

Several modeling nodes can be used to screen data to locate fields and records that are most likely to be of interest in modeling. This may be particularly useful in the preliminary stages of an analysis, in order to pare down the data while locating new patterns that may not be identified by existing models.



The Feature Selection node screens predictor fields for removal based on a set of criteria (such as the percentage of missing values); then it ranks the importance of remaining predictors relative to a specified target. For example, given a data set with hundreds of potential predictors, which are most likely to be useful in modeling patient outcomes?



The Anomaly Detection node identifies unusual cases, or outliers, that do not conform to patterns of “normal” data. With this node, it is possible to identify outliers even if they do not fit any previously known patterns and even if you are not sure exactly what you are looking for.

Data mining problems may involve hundreds, or even thousands, of fields that potentially may be used as predictors. As a result, a great deal of time and effort may be spent examining which fields or variables to include in the model. To narrow down the choices, the Feature Selection algorithm can be used to identify the fields that are most important for a given analysis. For example, if you are trying to predict

patient outcomes based on a number of factors, which factors are the most likely to be important?

Feature selection consists of three steps:

- **Screening.** Removes unimportant and problematic predictors and records or cases, such as predictors with too many missing values or predictors with too much or too little variation to be useful.
- **Ranking.** Sorts remaining predictors and assigns ranks based on importance.
- **Selecting.** Identifies the subset of features to use in subsequent models—for example, by preserving only the most important predictors and filtering or excluding all others.

In an age where many organizations are overloaded with too much data, the benefits of feature selection in simplifying and speeding the modeling process can be substantial. By focusing attention quickly on the fields that matter most, you can reduce the amount of computation required, more easily locate small but important relationships that might otherwise be overlooked, and, ultimately, obtain simpler, more accurate, and more easily explainable models. By reducing the number of fields used in the model, you may find that you can reduce the amount of data collected in future iterations and that scoring times may also be reduced.

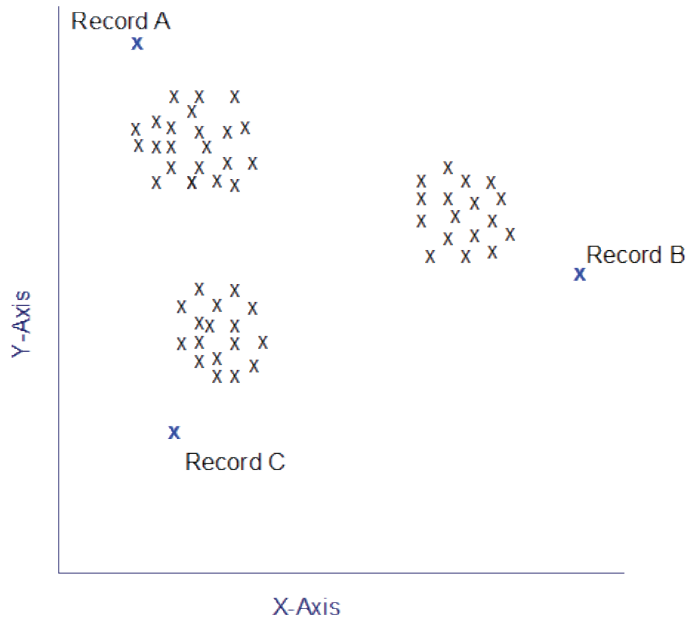
Anomaly Detection

Anomaly detection models are used to identify outliers, or “unusual” cases, in the data. Unlike other modeling methods that store rules about unusual cases, anomaly detection models store information on what “normal” behavior looks like. This makes it possible to identify outliers even if they do not conform to any known pattern, and it can be particularly useful in applications such as fraud detection, where new patterns may constantly be emerging. Anomaly detection is an unsupervised method, which means that it does not require a training data set containing known cases of fraud to use as a starting point.

While traditional methods of identifying outliers generally look at one or two variables at a time, anomaly detection can examine large numbers of fields to identify clusters or peer groups into which similar records fall. Each record can then be compared to others in its peer group to identify possible anomalies. The further away a case is from the “normal” center, the more likely it is to be unusual. For example, the

algorithm might lump records into three distinct clusters and flag those that fall far from the center of any one cluster.

Figure 4-1
Using clustering to identify potential anomalies



Each record is assigned an anomaly index, which is the ratio of the group deviation index to its average over the cluster that the case belongs to. The larger the value of this index, the more deviation the case has than the average. Under the usual circumstance, cases with anomaly index values less than 1 or even 1.5 would not be considered as anomalies because the deviation is just about the same or a bit more than the average. However, cases with an index value greater than 2 could be good anomaly candidates because the deviation is at least twice the average.

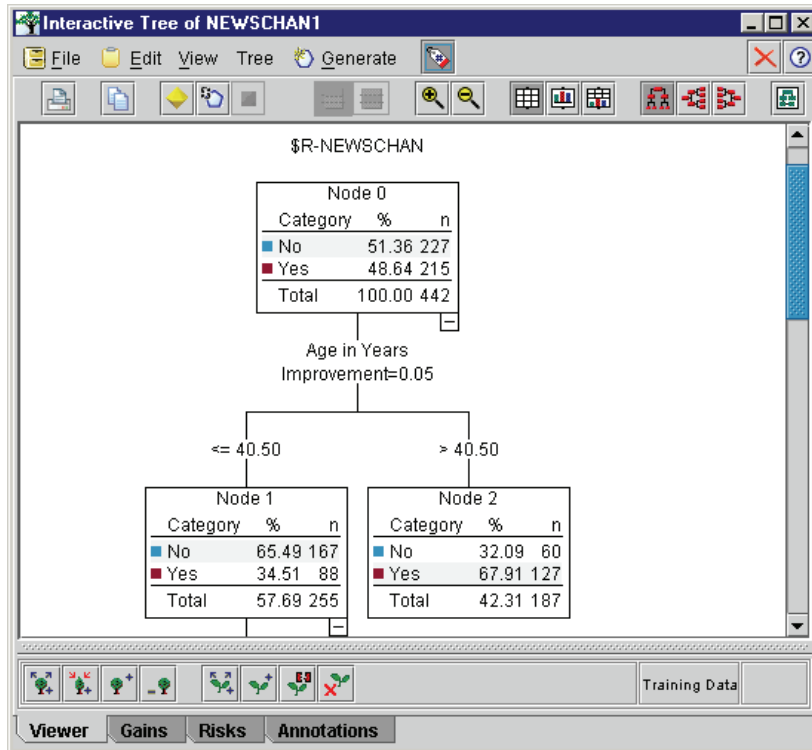
Anomaly detection is an exploratory method designed for a quick detection of unusual cases or records that should be candidates for further analysis. These should be regarded as *suspected* anomalies, which, on closer examination, may or may not turn out to be real. You may find that a record is perfectly valid but choose to screen it from the data for purposes of model building. Alternatively, if the algorithm repeatedly turns up false anomalies, this may point to an error or artifact in the data collection process.

Note that anomaly detection identifies unusual records or cases through cluster analysis based on the set of fields selected in the model without regard for any specific target (dependent) field and regardless of whether those fields are relevant to the pattern that you are trying to predict. For this reason, you may want to use anomaly detection in combination with feature selection or another technique for screening and ranking fields. For example, you can use feature selection to identify the most important fields relative to a specific target and then use anomaly detection to locate the records that are the most unusual with respect to those fields. (An alternative approach would be to build a decision tree model and then examine any misclassified records as potential anomalies. However, this method would be more difficult to replicate or automate on a large scale.)

Decision Trees (Rule Induction)

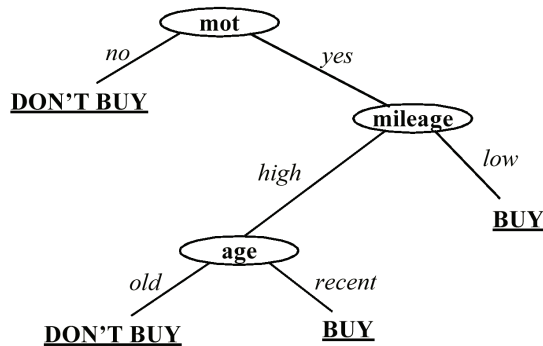
Decision tree models allow you to develop classification systems that predict or classify future observations based on a set of decision rules. If you have data divided into classes that interest you (for example, high- versus low-risk loans, subscribers versus nonsubscribers, voters versus nonvoters, or types of bacteria), you can use your data to build rules that you can use to classify old or new cases with maximum accuracy. For example, you might build a tree that classifies credit risk or purchase intent based on age and other factors.

Figure 4-2
Tree window



This approach, sometimes known as **rule induction**, has several advantages. First, the reasoning process behind the model is clearly evident when browsing the tree. This is in contrast to other “black box” modeling techniques in which the internal logic can be difficult to work out.

Figure 4-3
Simple decision tree



Second, 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 to reduce the data to relevant fields only before training another learning technique, such as a neural net.

Generated decision tree models 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'.
```

Tree-Building Algorithms

Four algorithms are available for performing classification and segmentation analysis. These algorithms all perform basically the same thing—they examine all of the fields of your database to find the one that gives the best classification or prediction by splitting the data into subgroups. The process is applied recursively, splitting subgroups into smaller and smaller units until the tree is finished (as defined by certain stopping criteria). The target and input fields used in tree building can be numeric ranges or

categorical, depending on the algorithm used. If a range target is used, a regression tree is generated; if a categorical target is used, a classification tree is generated.



The Classification and Regression Tree node generates a decision tree that allows you to predict or classify future observations. The method uses recursive partitioning to split the training records into segments by minimizing the impurity at each step, where a node is considered “pure” if 100% of cases in the node fall into a specific category of the target field. Target and predictor fields can be range or categorical; all splits are binary (only two subgroups).



The CHAID node generates decision trees using chi-square statistics to identify optimal splits. Unlike the C&RT and QUEST nodes, CHAID can generate non-binary trees, meaning that some splits have more than two branches. Target and predictor fields can be range or categorical. Exhaustive CHAID is a modification of CHAID that does a more thorough job of examining all possible splits but takes longer to compute.



The QUEST node provides a binary classification method for building decision trees, designed to reduce the processing time required for large C&RT analyses, while also reducing the tendency found in classification tree methods to favor predictors that allow more splits. Predictor fields can be numeric ranges, but the target field must be categorical. All splits are binary.



The C5.0 node builds either a decision tree or a ruleset. The model works by splitting the sample based on the field that provides the maximum information gain at each level. The target field must be categorical. Multiple splits into more than two subgroups are allowed.

General Uses of Tree-Based Analysis

The following are some general uses of tree-based analysis:

Segmentation. Identify persons who are likely to be members of a particular class.

Stratification. Assign cases into one of several categories, such as high-, medium-, and low-risk groups.

Prediction. Create rules and use them to predict future events. Prediction can also mean attempts to relate predictive attributes to values of a continuous variable.

Data reduction and variable screening. Select a useful subset of predictors from a large set of variables for use in building a formal parametric model.

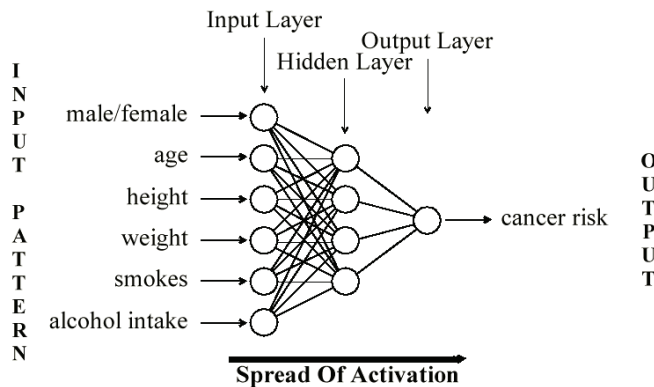
Interaction identification. Identify relationships that pertain only to specific subgroups and specify these in a formal parametric model.

Category merging and banding continuous variables. Recode group predictor categories and continuous variables with minimal loss of information.

Neural Networks

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

Figure 4-4
Structure of a neural network



A **neural network**, sometimes called a **multilayer 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**). Input data are presented to the first layer, and values are propagated from each neuron to every neuron in the next layer. Eventually, a result is delivered from the output layer.

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.

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.

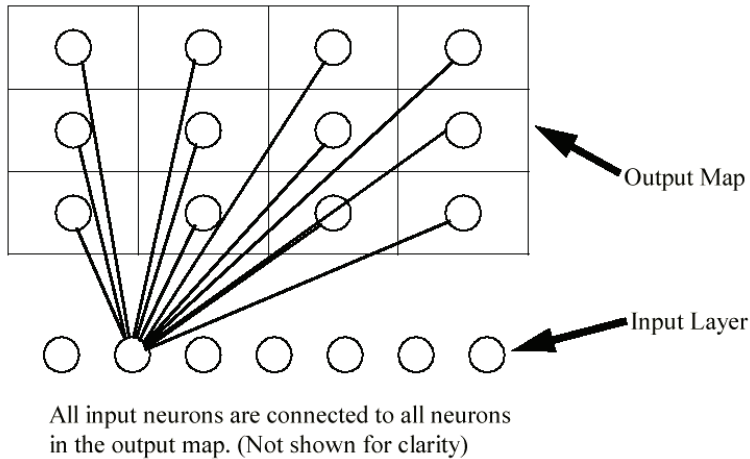
Kohonen Networks

Kohonen networks are a type of neural network that perform clustering, also known as 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. 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.

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. During training, each unit competes with all of the others to "win" each record.

The output map is a two-dimensional grid of neurons, with no connections between the units. A 3×4 map is shown below, although maps are typically larger than this.

Figure 4-5
Structure of a Kohonen network



Input data is presented to the input layer, and the values are propagated to the output layer. The output neuron with the strongest response is said to be the **winner** and is the answer for that input.

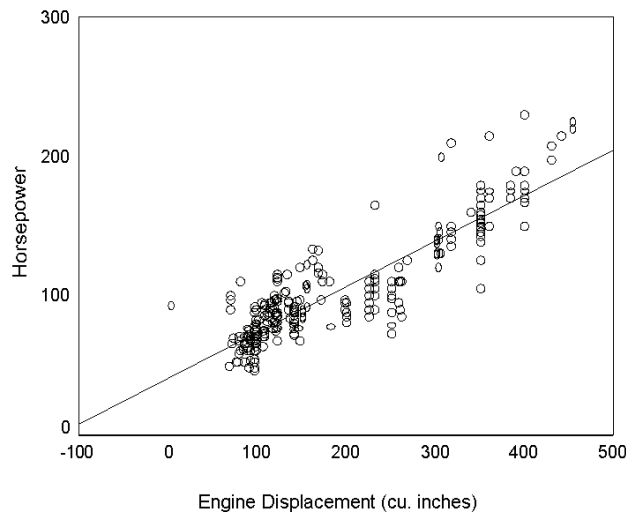
Initially, all weights are random. When a unit wins a record, its weights (along with those of other nearby units, collectively referred to as a **neighborhood**) are adjusted to better match the pattern of predictor values for that record. All of the input records are shown, and weights are updated accordingly. This process is repeated many times until the changes become very small. 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**).

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.

Statistical Models

Statistical models use mathematical equations to encode information extracted from the data. Several statistical modeling nodes are available.

Figure 4-6
Simple linear regression equation



Linear regression is a common statistical technique for summarizing data and making predictions by fitting a straight line or surface that minimizes the discrepancies between predicted and actual output values.



Logistic regression is a statistical technique for classifying records based on values of input fields. It is analogous to linear regression but takes a categorical target field instead of a numeric range.



The Factor/PCA node provides powerful data-reduction techniques to reduce the complexity of your data. 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. Factor analysis attempts to identify underlying factors that explain the pattern of correlations within a set of observed fields. 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.

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

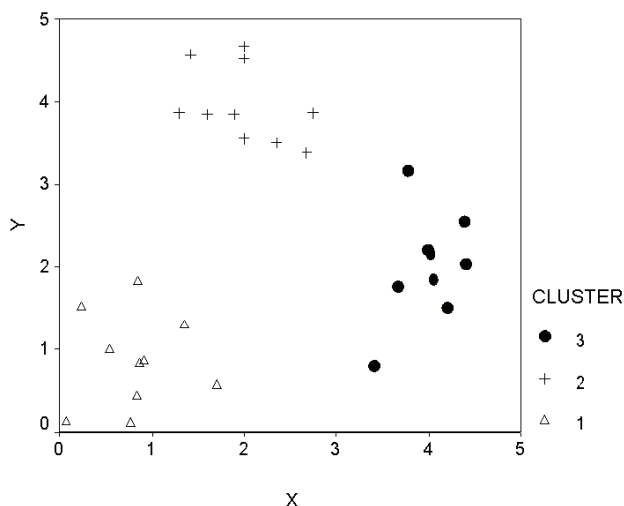
as neural networks) can ultimately give better results, you can use statistical models as baseline predictive models to judge the performance of advanced techniques.

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.

Figure 4-7
Simple clustering model



Clementine provides three methods for clustering:



The K-Means node clusters the data set into distinct groups (or clusters). The method defines a fixed number of clusters, iteratively assigns records to clusters, and adjusts the cluster **centers** until further refinement can no longer improve the model. Instead of trying to predict an outcome, K-Means uses a process known as unsupervised learning to uncover patterns in the set of input fields.



The TwoStep node uses a two-step clustering method. The first step makes a single pass through the data to compress 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. TwoStep has the advantage of automatically estimating the optimal number of clusters for the training data. It can handle mixed field types and large data sets efficiently.



The Kohonen node generates a type of neural network that can be used to cluster the data set into distinct groups. When the network is fully trained, records that are similar should appear close together on the output map, while records that are different will appear far apart. You can look at the number of observations captured by each unit in the generated model to identify the strong units. This may give you a sense of the appropriate number of clusters.

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.

Association Rules

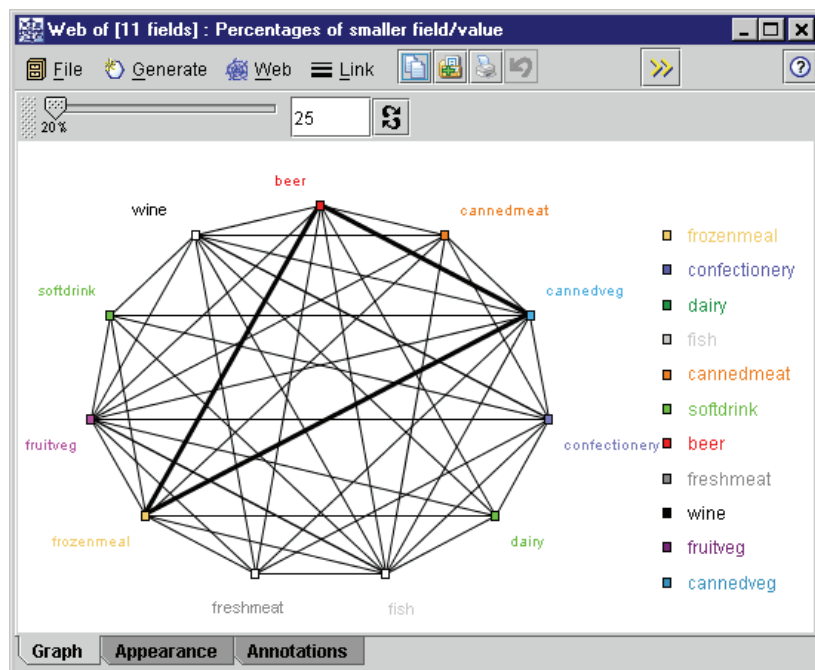
Association rules associate a particular conclusion (the purchase of a particular product) with a set of conditions (the purchase of several other products). For example, the rule

beer <= cannedveg & frozenmeal (173, 17.0%, 0.84)

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 4-8

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:



The Generalized Rule Induction (GRI) node discovers association rules in the data. For example, customers who purchase razors and aftershave lotion are also likely to purchase shaving cream. GRI extracts rules with the highest information content based on an index that takes both the generality (support) and accuracy (confidence) of rules into account. GRI can handle numeric and categorical inputs, but the target must be categorical.



The Apriori node extracts a set of rules from the data, pulling out the rules with the highest information content. Apriori offers five different methods of selecting rules and uses a sophisticated indexing scheme to process large data sets efficiently. For large problems, Apriori is generally faster to train than GRI; it has no arbitrary limit on the number of rules that can be retained, and it can handle rules with up to 32 preconditions. Apriori requires that input and output fields all be categorical but delivers better performance because it is optimized for this type of data.



The Sequence node discovers association rules in sequential or time-oriented data. A **sequence** is a list of item sets that tend to occur in a predictable order. For example, a customer who purchases a razor and aftershave lotion may purchase shaving cream the next time he shops. The Sequence node is based on the CARMA association rules algorithm, which uses an efficient two-pass method for finding sequences.

Text Mining

Text Mining for Clementine is an add-on product that uses linguistic methods to extract key concepts from text based on context. This information can be combined with existing structured data, such as demographics, and applied to modeling using Clementine's full suite of data mining tools to yield better and more focused decisions. A separate license is required.

Text Mining for Clementine uses the technology underlying LexiQuest Mine to process unstructured data. Based on 24 years of research in computational linguistics, LexiQuest Mine uses Natural Language Processing (NLP) to automate the process of reading documents to uncover their content. For example, in a 30-page publication, only two or three sentences may be relevant for your purposes. Instead of reading the entire 30 pages, you can automatically discover the main concepts. In addition, the underlying proprietary lexicon dictionaries allow the automatic classification of these concepts. As a result, you can quickly determine the relevance of the information to your needs.

A system that incorporates NLP can intelligently extract terms, including compound phrases. Moreover, knowledge of the underlying language allows classification of terms into related groups, such as products, organizations, or people, using the meaning and context of the text.

Linguistic systems are knowledge-sensitive—the more information contained in their dictionaries, the higher the quality of the results. Modification of the dictionary content, such as synonym definitions, can simplify the resulting information. This is often an iterative process and is necessary for accurate concept retrieval. Custom dictionaries for specific domains, such as CRM and genomics, are also included.

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, data entry 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 to try to identify the likely relevant factors that are not recorded. In trying to predict ice cream sales, for example, you may have a lot of information about retail outlets or sales history, but you may not have weather and temperature information, which 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 it to generate a full report.

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.

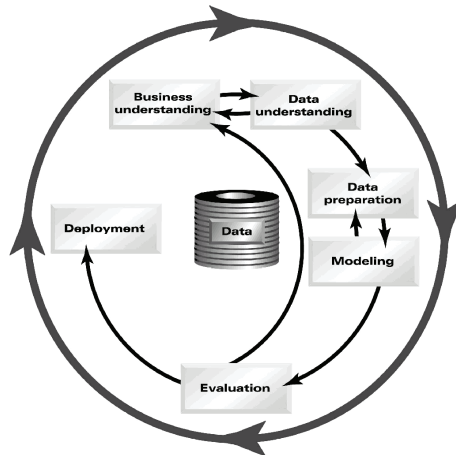
To stay on track, it helps to have an explicitly defined process model for data mining. The process model guides you through the critical issues outlined above and makes sure that the important points are addressed. It serves as a data mining road map so that you won't lose your way as you dig into the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

Figure 4-9
CRISP-DM process model



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.
- **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.
- **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 provides 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, forward, or backward 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, take smaller samples initially. 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.

Screening Fields for Analysis

Data mining problems may involve hundreds, or even thousands, of fields that potentially may be used as predictors. As a result, a great deal of time and effort may be spent examining which fields or variables to include in the model. To narrow down the choices, the Feature Selection algorithm can be used to identify the fields that are most important for a given analysis. For example, if you are trying to predict patient outcomes based on a number of factors, which factors are the most likely to be important?

Feature selection consists of three steps:

- **Screening.** Removes unimportant and problematic predictors and records or cases, such as predictors with too many missing values or predictors with too much or too little variation to be useful.
- **Ranking.** Sorts remaining predictors and assigns ranks based on importance.
- **Selecting.** Identifies the subset of features to use in subsequent models—for example, by preserving only the most important predictors and filtering or excluding all others.

In an age where many organizations are overloaded with too much data, the benefits of feature selection in simplifying and speeding the modeling process can be substantial. By focusing attention quickly on the fields that matter most, you can reduce the

amount of computation required, more easily locate small but important relationships that might otherwise be overlooked, and, ultimately, obtain simpler, more accurate, and more easily explainable models. By reducing the number of fields used in the model, you may find that you can reduce the amount of data collected in future iterations and that scoring times may also be reduced.

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.

Data Mining Examples

The best way to learn about data mining in practice is to start with an example. Several short application examples are included as an appendix to the *Clementine User's Guide*. For more information, see "Application Examples" in Chapter 12 on p. 223.

Also available are full-sized application templates for a variety of fields, such as fraud detection, customer relationship management, and micro-array analysis. These examples are called **Clementine Application Templates** (CATs) and are available from your sales representative.

Using Clementine Application Templates

A Clementine Application Template (CAT) consists of:

- A library of Clementine streams, typically organized as a Clementine project (*.cpj*) file.
- Sample data that allows streams to be executed for illustrative purposes without modification. CAT data is supplied in flat files to avoid dependence on a database system. In the case of CATs that use the SPSS Reference Data model, data is delivered in tables preconfigured for a SQL Server database.
- A user's guide that explains the application, the approach and structure used in the stream library, the purpose and use of each stream, and how to apply the streams to new data.

Opening CAT Streams

Once you have purchased and installed a CAT, you can access the streams or project files easily from Clementine by using options from the File menu or a CAT toolbar button.

- ▶ On the toolbar at the top of the Clementine window, click the CAT button.

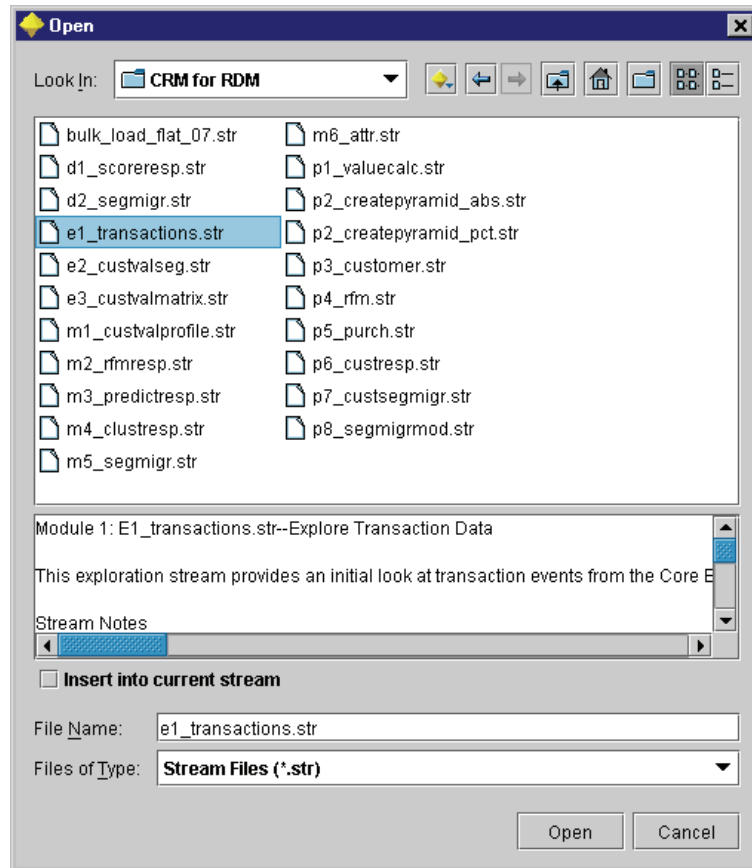
Figure 4-10

Toolbar button used to open the template library



- ▶ Alternatively, from the File menu, choose:
Template Library
- ▶ Using the dialog box, which opens to the templates directory, select the CAT stream that you want to open. You can either choose to open it as a separate stream or insert the CAT stream into the currently open stream.

Figure 4-11
Opening a CAT stream



Opening a CAT Project

If the CAT you purchased ships with a project (.cpj) file, you can use this project file to open and organize the many CAT streams.

To open a CAT project:

- From the File menu, choose:
 - Project
 - Open Project

- ▶ In the dialog box that appears, navigate to the application template directory using the diamond drop-down list.
- ▶ Alternatively, you can navigate manually to the directory. By default, CATs are installed in *C:\Program Files\Clementine\9.0\STL*.

For more information on CATs, see the documentation available on the CAT CD.

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.

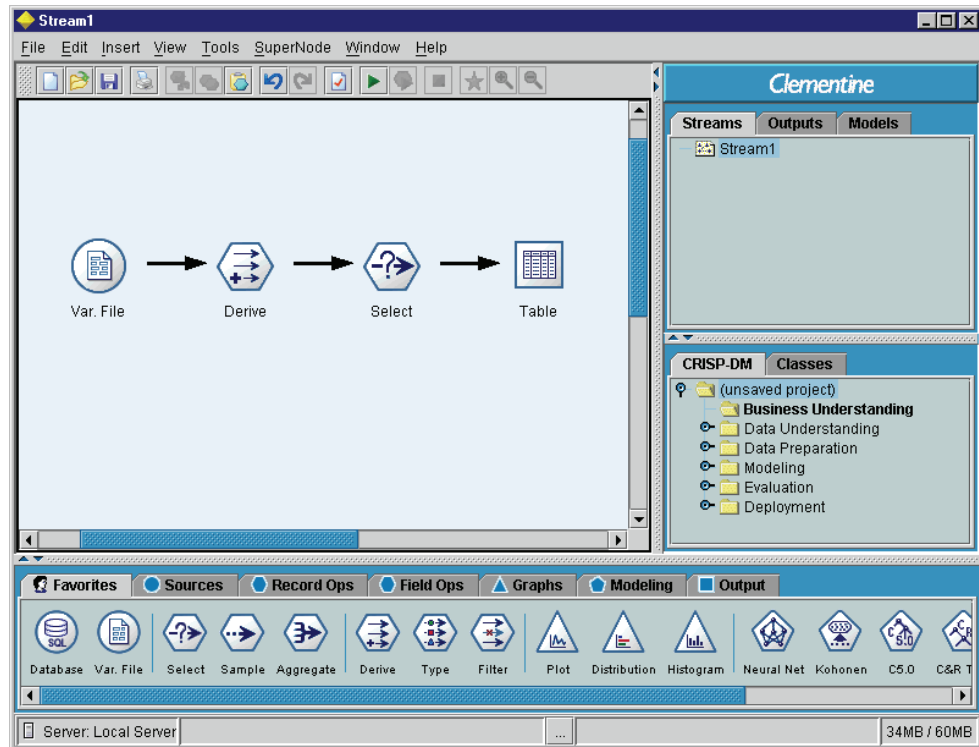
Building Data Streams

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 5-1

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 (in the *Demos* folder of your program installation), see Chapter 12.

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 5-2
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.

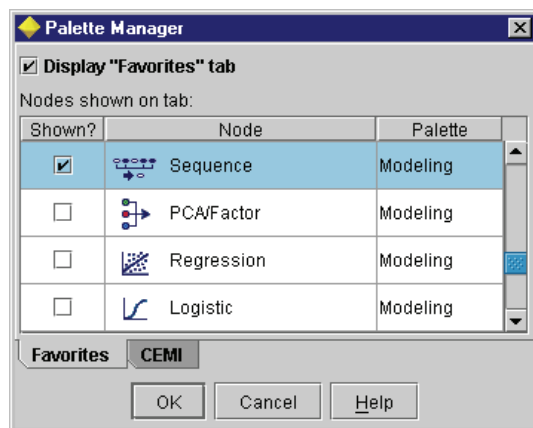
- **Modeling.** Nodes representing the powerful modeling algorithms available in Clementine, such as neural nets, decision trees, clustering algorithms, and data sequencing.
- **Output.** Nodes used to produce a variety of output for Clementine data, charts, and model results. Output can be viewed within 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, choose Favorites.

Figure 5-3
Selecting nodes to add to the Favorites tab



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).

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 p. 65.
- 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 available options 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.

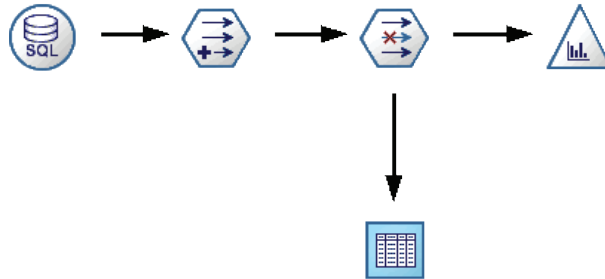
To add and connect nodes by double-clicking:

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 5-4

Stream created by double-clicking nodes from the palettes

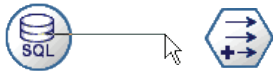


To connect nodes using the middle mouse button:

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 while dragging with the mouse from one node to another.)

Figure 5-5

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.

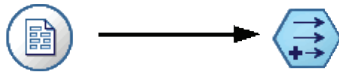
Figure 5-6

Connecting nodes using the Connect option from the context menu



Figure 5-7

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:

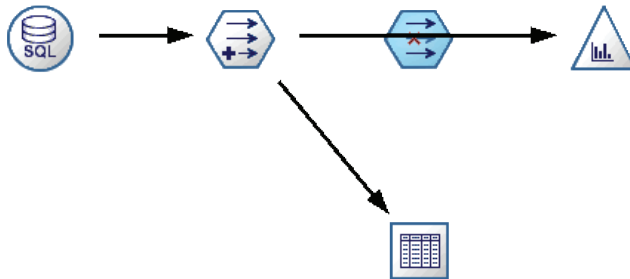
- A connection leading to a source node
- A connection leading from a terminal node
- A node having more than its maximum number of input connections
- Connecting two nodes that are already connected
- Circularity (data returns to a node from which it has already flowed)

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 5-8
Bypassing a previously connected Filter node



To bypass a node:

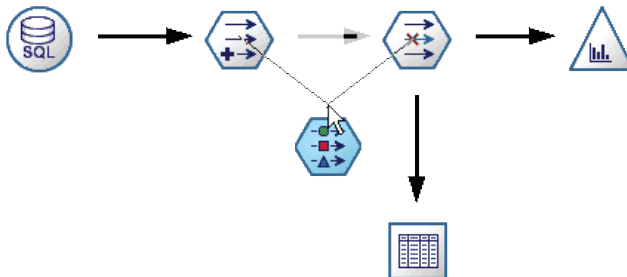
- On the stream canvas, use the middle mouse button to double-click the node that you want to bypass. Alternatively, you can use Alt-double-click.

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

Adding Nodes in Existing Connections

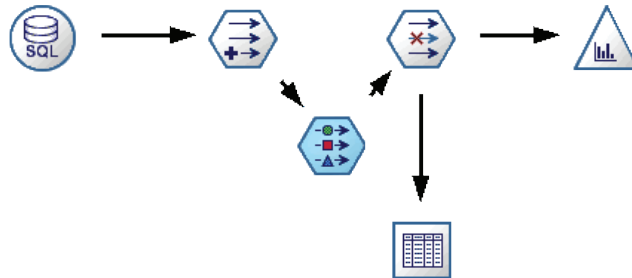
You can add a new node between two connected nodes by dragging the arrow that connects the two nodes.

Figure 5-9
Connecting a new node between two connected nodes



- ▶ With the middle mouse button, click and drag the connection arrow into which you want to insert the node. Alternatively, you can hold down the Alt-key while clicking and dragging to simulate a middle mouse button.

Figure 5-10
New stream



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

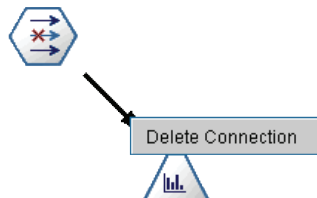
Note: You can remove new connections from the node and restore the original by **bypassing** the node.

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 arrowhead.
- ▶ From the context menu, select Delete Connection.

Figure 5-11
Deleting the connection between nodes in a stream



Or, you can delete a connection as follows:

- ▶ Select a node and press the F3 key to delete all connections.

- Select a node, and from the main menus choose:

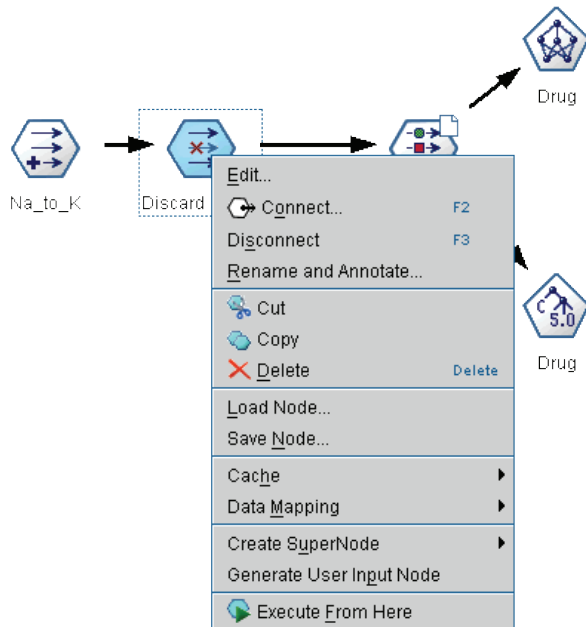
Edit
Node
Disconnect

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.

Figure 5-12

Context menu options for nodes



- 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 Annotations tab of the editing dialog box.
- 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 Copy to make a copy of the node with no connections. This can be added to a new or existing stream.
- 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.
- Select Generate User Input Node to replace the selected node. Examples generated by this node will have the same fields as the current node.
- Select Execute From Here to execute all terminal nodes downstream from the selected node.

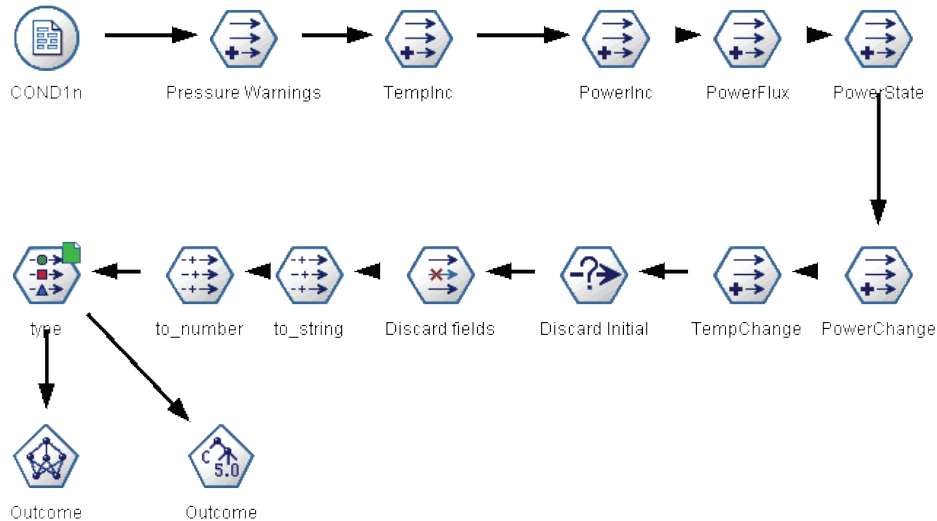
Caching Options for Nodes

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 from the data source.

For example, suppose that 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 on 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.

Figure 5-13
Caching at the Type node to store newly derived fields



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.

Caching Nodes in the Database

For streams executed in the database, data can be cached midstream to a temporary table in the database rather than the file system. When combined with SQL optimization, this may result in significant gains in performance. For example, the output from a stream that merges multiple tables to create a data mining view may be cached and reused as needed. By automatically generating SQL for all downstream nodes, performance can be further improved.

To take advantage of database caching, both SQL optimization and database caching must be enabled. Note that Server optimization settings override those on the Client. For more information, see “Setting Optimization Options” in Chapter 3 on p. 27.

With database caching enabled, simply right-click on any non-terminal node to cache data at that point, and the cache will be created automatically directly in the database the next time the stream is executed. If database caching or SQL optimization is not enabled, the cache will be written to the file system instead.

Note: The following databases support temporary tables for the purpose of caching: DB2, Netezza, Oracle, SQL Server, and Teradata. Other databases will use a normal table for database caching. The SQL code can be customized for specific databases by editing properties in the relevant configuration file—for example, *c:\program files\SPSS Clementine\10.0\config\odbc-teradata-properties.cfg*. For more information, see the comments in the default configuration file, *odbc-properties.cfg*, installed in the same folder.

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 and Streams

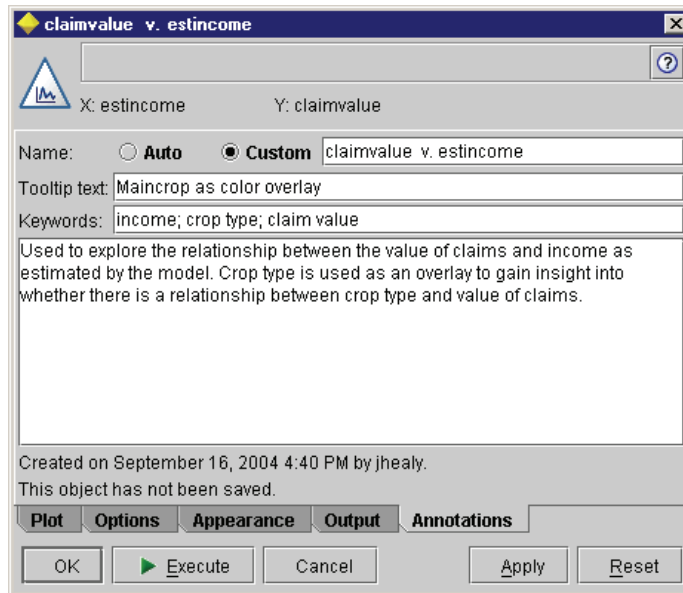
Nodes, streams, and models can be annotated in a number of ways. You can add descriptive annotations and specify a custom name. These options are useful especially when generating reports for streams added to the projects tool. For nodes, you can also add ToolTip text to help distinguish between similar nodes on the stream canvas.

Adding Annotations

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.

- ▶ To annotate a node, right-click on the node on the stream canvas, and select Rename and Annotate. The editing dialog box opens with the Annotations tab visible.
- ▶ To annotate a stream, select Stream Properties from the Tools menu. (Alternatively, you can right-click a stream in the managers window and select Stream Properties.)

Figure 5-14
Annotations tab options



Name. Select Custom to adjust the autogenerated name or to create a unique name for the node as displayed on the stream canvas.

ToolTip text. Enter text used as a ToolTip on the stream canvas. This is particularly useful when working with a large number of similar nodes.

Keywords. Specify keywords to be used in project reports and when searching or tracking objects stored in the Predictive Enterprise Repository. (For more information, see “Predictive Enterprise Repository” in Chapter 11 on p. 203.) Multiple keywords can be separated by semicolons—for example, income; crop type; claim value. White spaces at the beginning and end of each keyword are trimmed—for example, income ; crop type will produce the same results as income;crop type. (White spaces within keywords are not trimmed, however. For example, crop type with one space and crop type with two spaces are not the same.)

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

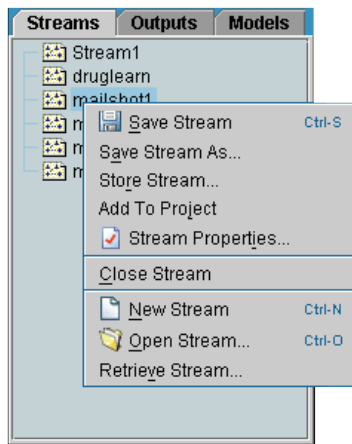
information with the node. You can also choose to include these annotations in a project report created with the projects tool.

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 5-15
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.
- Store and retrieve streams from the Predictive Enterprise Repository (if available at your site). For more information, see “Predictive Enterprise Repository” in Chapter 11 on p. 203.

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.

Figure 5-16
Options tab in stream properties dialog box

Stream1

Calculations in: ☒ Radians ☐ Degrees

Import date/time as: ☒ Date/Time ☐ String

Date format: YYYY-MM-DD

Time format: HH:MM:SS ☐ Rollover days/mins

Number display format: Standard (###.###)

Standard decimal places: 3

Scientific decimal places: 3

Currency decimal places: 2

Decimal symbol: Period (.)

Grouping symbol: None

Date baseline (1st Jan): 1900

2-digit dates start from: 1930

Encoding: Server default

☒ Maximum set size 250

☒ Limit set size for Neural, Kohonen and K-Means modeling 20

Ruleset Evaluation: Voting

☐ Refresh source nodes on execution

☐ Display field and value labels in output

Save As Default

Options Layout

Messages Parameters Script Globals Annotations

OK Cancel Apply Reset

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.

Rollover days/mins. For time formats, select whether negative time differences should be interpreted as referring to the previous day or hour.

Number display format. You can choose from standard (####.###), scientific (#.###E+##), or currency display formats (\$###.##).

Decimal places (standard, scientific, currency). For number display formats, specifies the number of decimal places to be used when displaying or printing real numbers. This option is specified separately for each display format.

Decimal symbol. Select either a comma (,) or a period (.) as a decimal separator.

Grouping symbol. For number display formats, select the symbol used to group values (for example, the comma in 3,000.00). Options include none, period, comma, space, and locale-defined (in which case the default for the current locale is used).

Date baseline (1st Jan). Select the baseline years (always January 1) 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.

Encoding. Specify the encoding method used (System default or UTF-8).

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 that 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.

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) 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.

Display field and value labels in output. Displays field and value labels in tables, charts, and other output. If labels don't exist, the field names and data values will be displayed instead. Labels are turned off by default; however, you can toggle labels on an individual basis elsewhere in Clementine. You can also choose to display labels on the output window using a toggle button available on the toolbar.

Figure 5-17

Toolbar icon used to toggle field and value labels



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.

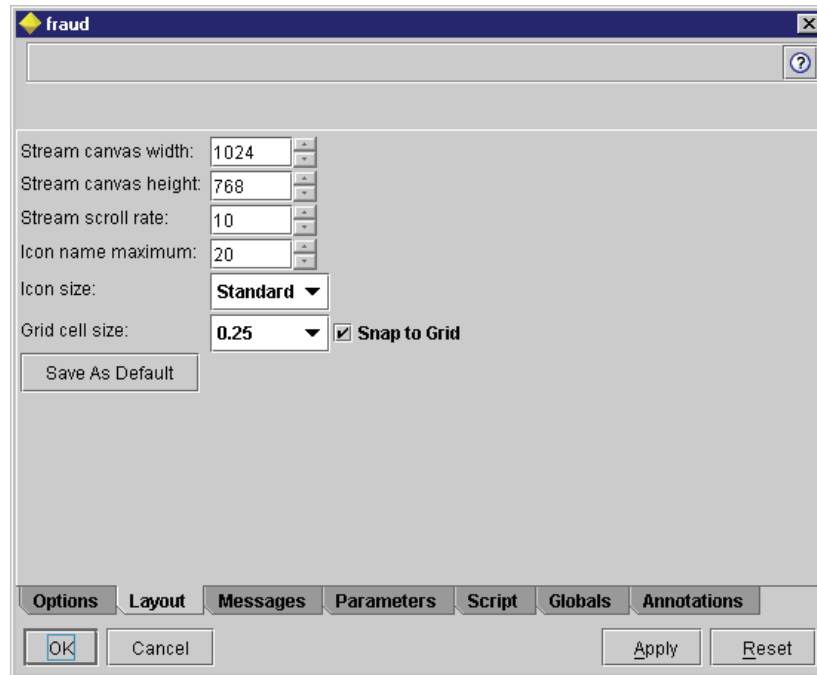
Setting Options for Stream Layout

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:

- ▶ From the File menu, choose Stream Properties. Alternatively, from the Tools menu, choose:
Stream Properties
Layout
- ▶ Click the Layout tab in the stream properties dialog box.

Figure 5-18
Layout tab in 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 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.

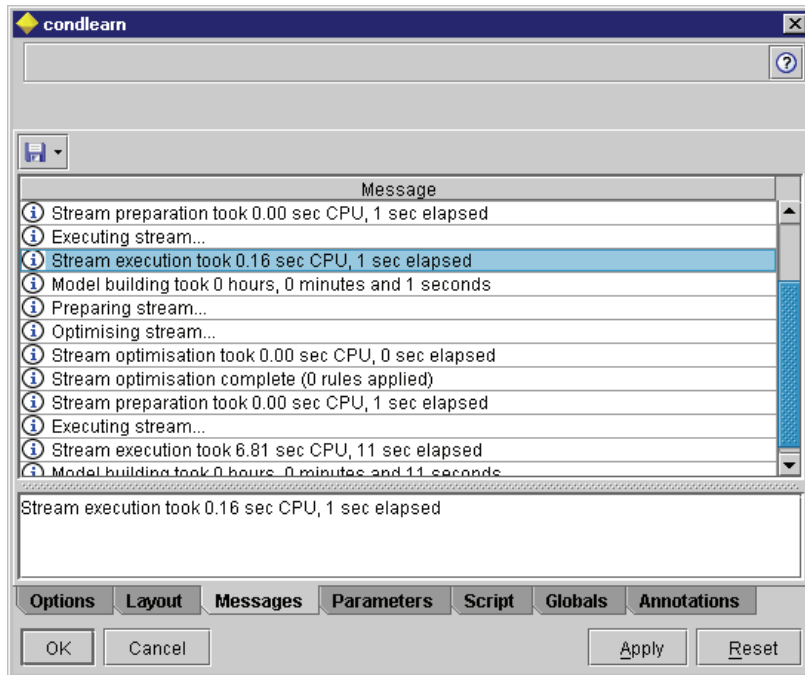
Viewing Stream Execution Messages

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

To view stream messages:

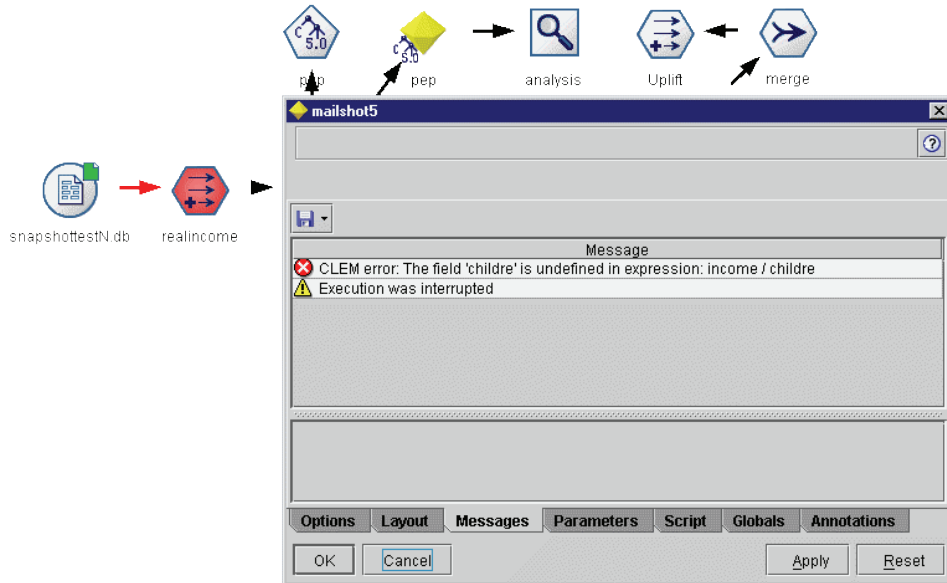
- ▶ From the File menu, choose Stream Properties. Alternatively, from the Tools menu, choose:
Stream
Messages
- ▶ Click the Messages tab in the stream properties dialog box.

Figure 5-19
Messages tab in 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.

Figure 5-20
Stream execution with errors reported



If SQL optimization and logging options are enabled in the User Options dialog box, then information on generated SQL is also displayed. For more information, see “Setting Optimization Options” in Chapter 3 on p. 27.

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.

Setting Stream and Session Parameters

Parameters can be defined in Clementine for use in CLEM expressions and in scripting. They are, in effect, user-defined variables that are saved and persisted with the current stream, session, or SuperNode, and they can be accessed from the user interface as well as through scripting. If you save a stream, for example, any parameters set for that stream are also saved. (This distinguishes them from local script variables, which can be used only in the script where they are declared.) Parameters are often used in scripting as part of a CLEM expression in which the parameter value is specified in the script.

The scope of a parameter depends on where it is set:

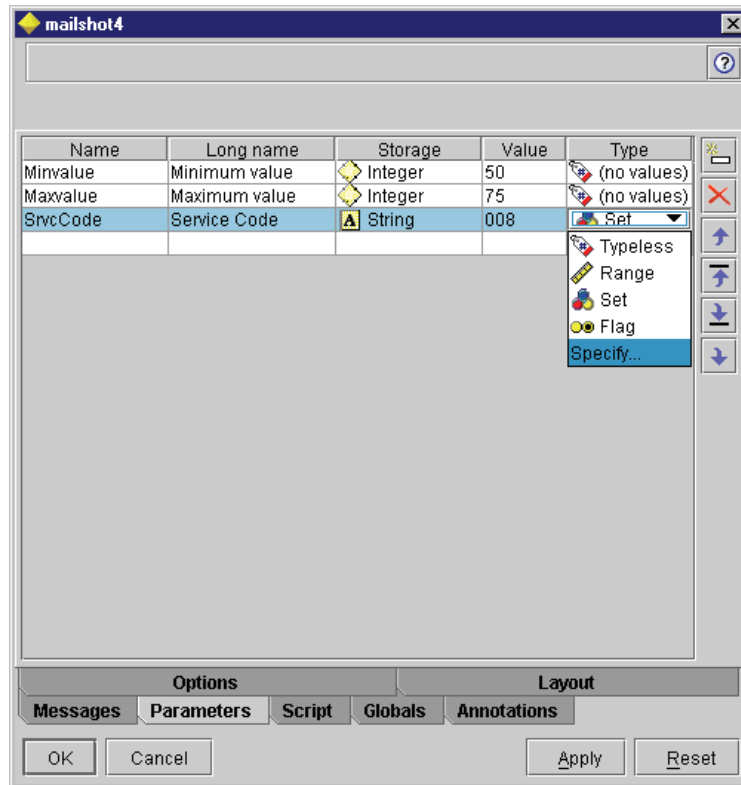
- **Stream parameters** can be set in a stream script or in the stream properties dialog box, and they are available to all nodes in the stream. They will appear on the Parameters drop-down list in the Expression Builder.
- **Session parameters** can be set in a stand-alone script or in the session parameters dialog box. They are available to all streams used in a single instance of Clementine (all streams listed on the Streams tab in the managers window).

Parameters can also be set for SuperNodes, in which case they are visible only to nodes encapsulated within that SuperNode.

To set stream and session parameters through the user interface:

- ▶ To set stream parameters, from the menus choose:
 - Tools
 - Stream Properties
 - Parameters
- ▶ To set session parameters, choose Set Session Parameters from the Tools menu.

Figure 5-21
Setting parameters for streams



Name. Parameter names are listed here. You can create a new parameter by entering a name in this field. For example, to create a parameter for the minimum temperature, you could type minvalue. Do not include the \$P- prefix that denotes a parameter in CLEM expressions. This name is also 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. For date parameters, note that values must be specified using ISO standard notation as below.

Value. Lists the current value for each parameter. Adjust the parameter as desired. Note that for date parameters, values must be specified in ISO standard notation (that is, YYYY-MM-DD). Dates specified in other formats are not accepted.

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.

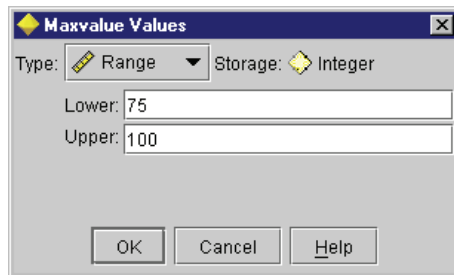
Note that long name, storage, and type options can be set for parameters through the user interface only. These options cannot be set using scripts.

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.

Specifying Values for a Parameter Type

You can make values for a parameter available during stream deployment to an external application, such as a Web analytics package that reads data modeling streams. This dialog box allows you to specify the values available to an external user executing the stream. Depending on the data type, value constraints vary dynamically in the dialog box. The options shown here are identical to the options available for values from the Type node.

Figure 5-22
Specifying available values for a parameter



Type. Displays the currently selected type. You can change the type to reflect the way that you intend to use the parameter in Clementine.

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. You can alter storage type on the main Parameters tab.

The bottom half of the dialog box dynamically changes depending on the type selected above.

Range Types

Lower. Specify a lower limit for the parameter values.

Upper. Specify an upper limit for the parameter values.

Set Types

Values. This option allows you to specify values for a parameter that will be used as a set field. Values will not be coerced in the Clementine stream but will be used in a drop-down list for external deployment applications. Using the arrow and delete buttons, you can modify existing values as well as reorder or delete values.

Flag Types

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

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

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. For more information, see “Annotating Nodes and Streams” on p. 74.

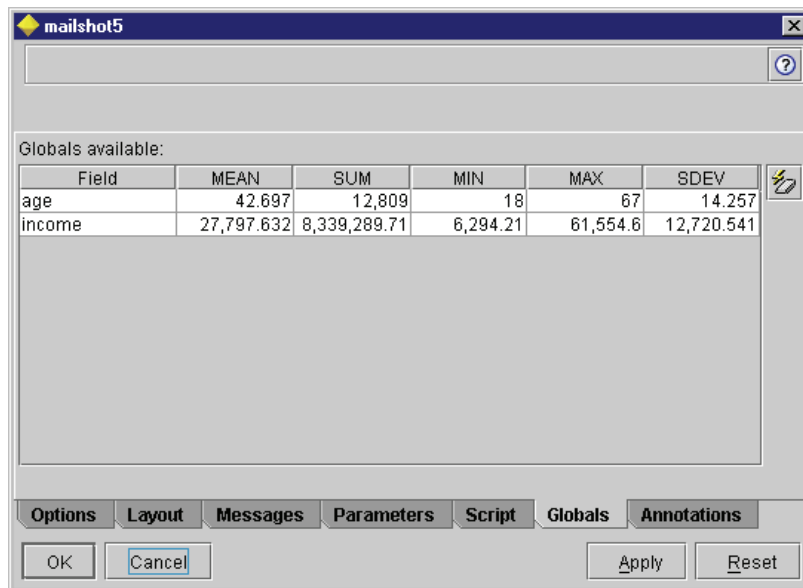
Viewing Global Values for Streams

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. For more information, see “Global Functions” in Chapter 8 on p. 161.

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.

Figure 5-23*Viewing global values available for the stream*

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 3 on p. 15.
- 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.

Automatic backup files. Each time a stream is saved, the previously saved version of the file is automatically preserved as a backup, with a hyphen appended to the filename (for example *mystream.str-*). To restore the backed-up version, simply delete the hyphen and reopen the file.

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
 - Save State or Save State As
- ▶ 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.

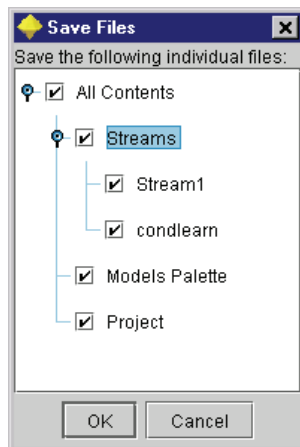
Saving Nodes

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 5-24
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.

Saving Output

Tables, graphs, and reports generated from Clementine output nodes can be saved in output object (*.cou) format.

- ▶ When viewing the output you want to save, from the output window menus choose:
File
Save
- ▶ Specify a name and location for the output file.
- ▶ Optionally, select Add file to project in the Save dialog box to include the file in the current project. For more information, see “Introduction to Projects” in Chapter 9 on p. 165.

Alternatively, you can right-click on any output object listed in the managers window (upper right corner of the Clementine window) and select Save from the context menu.

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.

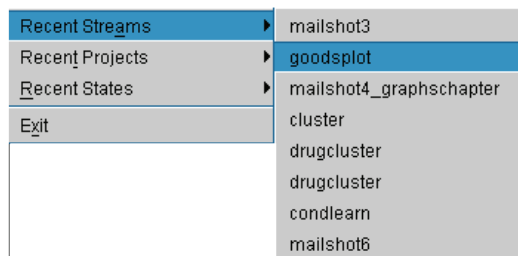
- The Cache File node has been replaced by the SPSS Import node. Any streams that you load containing Cache File nodes will be replaced by SPSS Import nodes.
- The Build Rule node has been replaced by the C&R Tree node. Any streams that you load containing Build Rule nodes will be replaced by C&R Tree 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 5-25

Opening recently used options from the File menu



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.

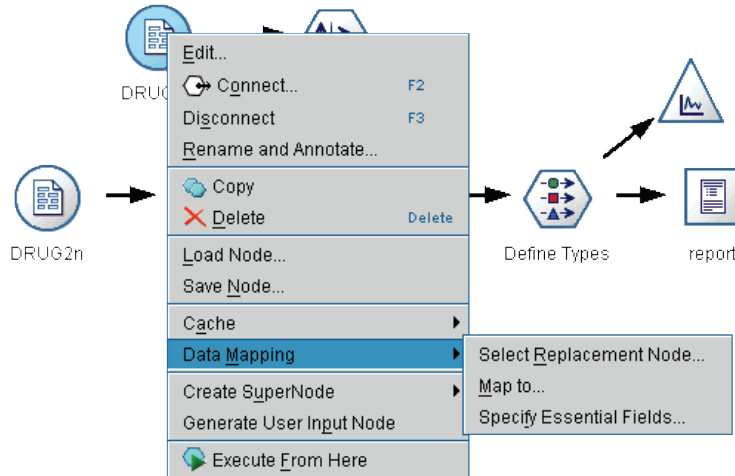
The data mapping tool allows you to join together two stream fragments and be sure that all of the (essential) field names match up properly. A common use is to replace a source node defined in a Clementine Application Template (CAT) with a source node that defines your own data set. In essence, mapping 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 method 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 method 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.

Figure 5-26
Selecting data mapping options



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

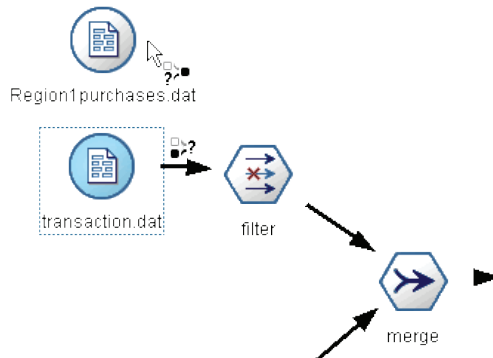
To replace the data source for a template stream with a new source node bringing your own data into Clementine, you should use the Select Replacement Node option from the Data Mapping context menu option. This option is available for all nodes except Merge, Aggregate, and all terminal nodes. Using the 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 p. 98.

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 option on the context menu for the template source node, choose Select Replacement Node. Then select the source node for the replacement data.

Figure 5-27
Selecting a replacement source node

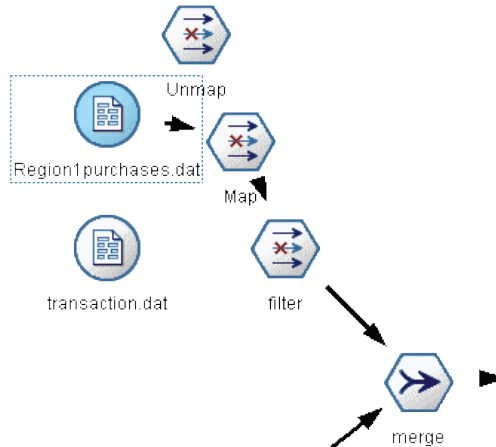


Step 4: Check mapped fields. In the dialog box that opens, check that the software is mapping fields properly from the replacement data source to the stream. Any unmapped essential fields are displayed in red. These fields are used in stream operations and must be replaced with a similar field in the new data source in order for downstream operations to function properly. For more information, see “Examining Mapped Fields” on p. 99.

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 5-28

New data source successfully mapped to the template stream

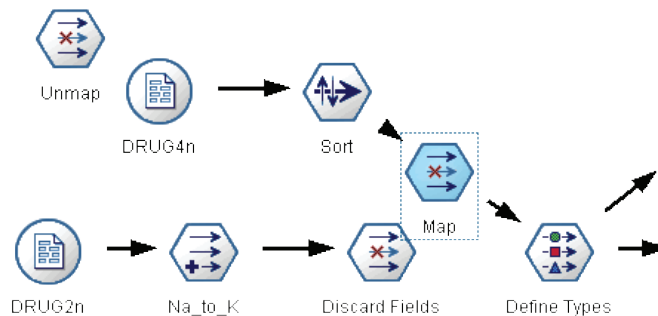


Mapping between Streams

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 Map to from the Data Mapping context menu. 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 5-29

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, choose:
 - 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

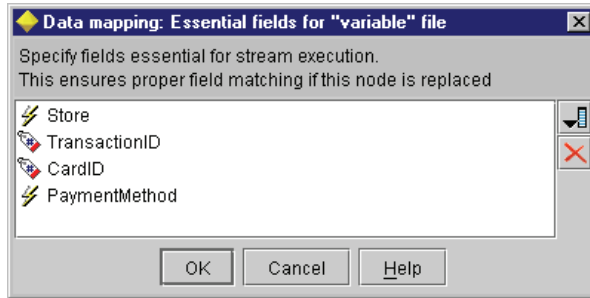
When mapping to a template, essential fields will typically be specified by the template author. These essential fields indicate whether a particular field is used in downstream operations. For example, the existing stream may build a model that uses a field called *Churn*. In this stream, *Churn* is 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.

Figure 5-30
Specifying essential fields

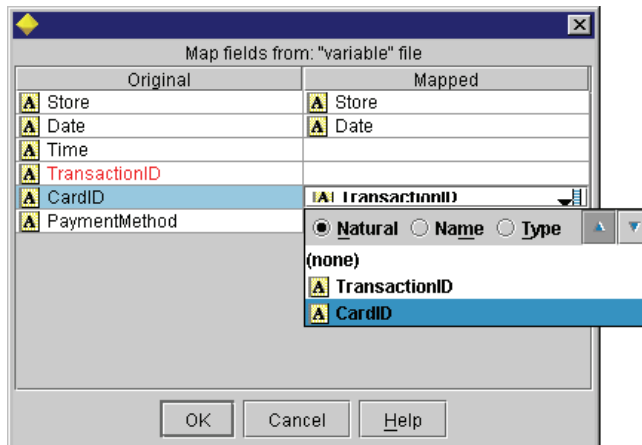


- Using the Field Chooser, you can add or remove fields from the list. To open the Field Chooser, click the icon to the right of the fields list.

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.

Figure 5-31
Selecting fields for mapping



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.

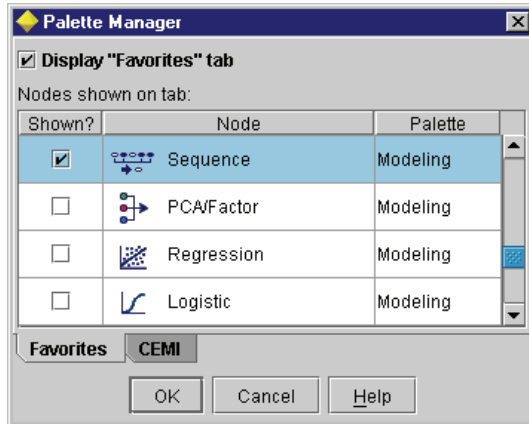
If you are unsure of which fields to map, it may be useful to examine the source data closely before mapping. For example, you can use the Types tab in the source node to review a summary of the source data.

Tips and Shortcuts

Work quickly and easily in Clementine by familiarizing yourself with the following shortcuts and tips:

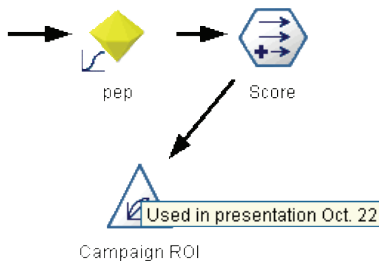
- **Build streams quickly by double-clicking.** Simply double-click a node on the palette to add and connect it to the current stream.
- **Use key combinations to select downstream nodes.** Press Ctrl-Q and Ctrl-W to toggle the selection of all nodes downstream.
- **Use shortcut keys to connect and disconnect nodes.** When a node is selected in the canvas, press F2 to begin a connection, press Tab to move to the desired node, and press the spacebar to complete the connection. Press F3 to disconnect all inputs and outputs to the selected node.
- **Customize the Favorites tab with your favorite nodes.** From the Tools menu, choose Favorites to open a dialog box for adding and removing nodes from the Favorites tab.

Figure 5-32
Favorites manager



- **Rename nodes and add ToolTips.** Each node dialog box includes an Annotations tab on which you can specify a custom name for nodes on the canvas as well as add ToolTips to help organize your stream. You can also include lengthy annotations to track progress, save process details, and denote any business decisions required or achieved.

Figure 5-33
ToolTip and custom node name



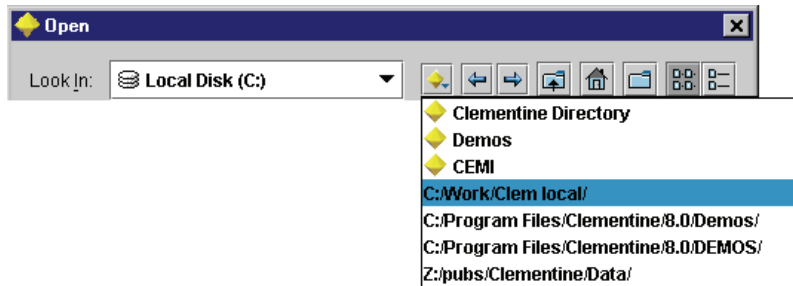
- **Insert values automatically into a CLEM expression.** Using the Expression Builder, accessible from a variety of dialog boxes (such as those for Derive and Filler nodes), you can automatically insert field values into a CLEM expression. Click the values button on the Expression Builder to choose from existing field values.

Figure 5-34
Values button



- **Browse for files quickly.** When browsing for files, use the File drop-down list (yellow diamond button) to access previously used directories as well as Clementine default directories. Use the forward and back buttons to scroll through accessed directories.

Figure 5-35
File-browsing options



- **Minimize output window clutter.** You can close and delete output quickly using the red X button at the top right corner of all output windows. This enables you to keep only promising or interesting results on the Outputs tab of the managers window.

A full range of keyboard shortcuts is available for the software. For more information, see “Keyboard Accessibility” in Appendix A on p. 282.

Did you know that you can...

- Drag and select a group of nodes on the stream canvas using your mouse.
- Copy and paste nodes from one stream to another.
- Access Help from every dialog box and output window.
- Get Help on CRISP-DM, the Cross-Industry Standard Process for Data Mining. (From the Help menu, choose CRISP-DM Help.)

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, the field *Sex* should contain the values *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 non-response. At times, you may want to examine these blanks more closely to determine whether a non-response, 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, resulting in less-accurate models.

There are two types of missing values in Clementine:

- **System-missing values.** Also called **nulls**, these are non-string values that have been left blank in the database or source file and have not been specifically defined as “missing” in a source or Type node. System-missing values are displayed as \$null\$ in Clementine. Note that empty strings are not considered nulls in Clementine, although they may be treated as nulls by certain databases (see below).
- **User-defined missing values.** Also called **blanks**, these are values, such as unknown, 99, or -1, that are explicitly defined in a source node or Type node as missing. Optionally, you can also choose to treat nulls and white space as blanks, which allows them to be flagged for special treatment and to be excluded from most

calculations. For example, you can use the @BLANK function to treat these values along with other types of missing values.

Figure 6-1
Specifying missing values for a range variable

Empty strings and white space. Clementine treats empty string values and white space (strings with no visible characters) as distinct from null values. Empty strings are treated as equivalent to white space for most purposes; for example, if you select the option to treat white space as blanks in a source or Type node, this setting applies to empty strings as well. However, empty strings may be handled differently than other white space by certain databases and by the Quality node.

Reading in mixed data. Note that when you are reading in fields with numeric storage (either integer, real, time, timestamp, or date), any non-numeric values are set to *null* or *system missing*. This is because, unlike some applications, Clementine does not allow mixed storage types within a field. To avoid this, any fields with mixed data should be read in as strings by changing the storage type in the source node or external application as necessary.

Reading empty strings from Oracle. When reading from or writing to an Oracle database, be aware that, unlike Clementine and unlike most other databases, Oracle treats and stores empty string values as equivalent to null values. This means that the same data extracted from an Oracle database may behave differently than when extracted from a file or another database, and the data may return different results.

Treating Missing Values

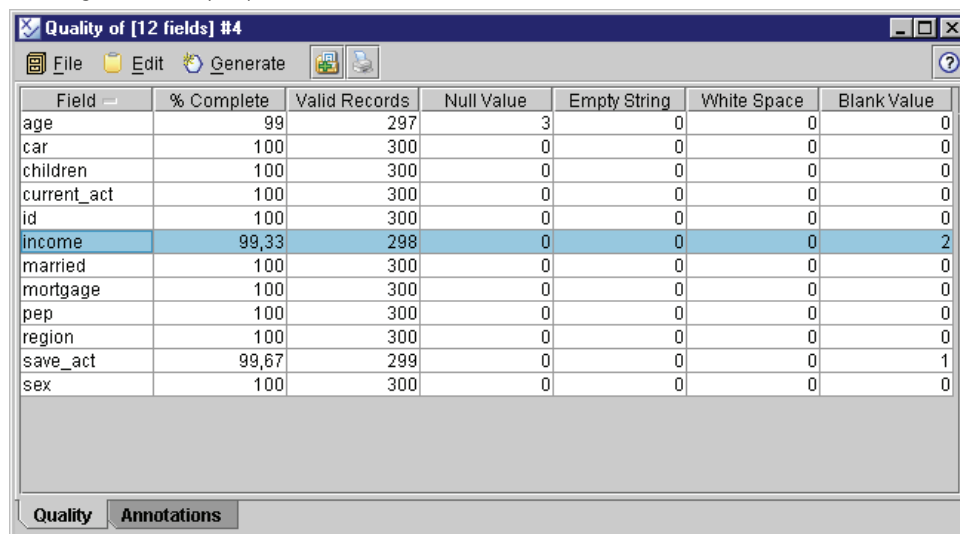
You should decide how to treat missing values in light of your business or domain knowledge. In order to ease training time and increase accuracy, you may want to remove blanks from your data set. On the other hand, the presence of blank values may lead to new business opportunities or additional insight. In choosing the best technique, you should consider the following aspects of your data:

- Size of the data set
- Number of fields containing blanks
- Amount of missing information

Generating a Quality Report

Using a Quality node, you can generate a report that lists the number of missing values. Once you have examined the report, you can use this node to automate the selection and filtering of records and fields with missing values.

Figure 6-2
Viewing the *Quality report for a data set*



Field	% Complete	Valid Records	Null Value	Empty String	White Space	Blank Value
age	99	297	3	0	0	0
car	100	300	0	0	0	0
children	100	300	0	0	0	0
current_act	100	300	0	0	0	0
id	100	300	0	0	0	0
income	99,33	298	0	0	0	2
married	100	300	0	0	0	0
mortgage	100	300	0	0	0	0
pep	100	300	0	0	0	0
region	100	300	0	0	0	0
save_act	99,67	299	0	0	0	1
sex	100	300	0	0	0	0

Once you have analyzed these factors, you can deal with missing values at the record or field level, as described in the sections that follow.

Treating Records 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 are 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

If the majority of missing values is concentrated in a small number of fields, you can address them at the field level rather than at the record level. This approach also allows you to experiment with the relative importance of particular fields before deciding on an approach for handling missing values. If a field is unimportant in modeling, it probably isn't worth keeping, regardless of how many missing values it has.

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.

Field Type

In determining which method to use, you should also consider the **type** of the field with missing values.

Numeric range fields. 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.

Categorical fields. For categorical fields, 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.

Screening or Removing Fields

To screen out fields with too many missing values, you have several options:

- You can use a Feature Selection node to screen out fields with more than a specified percentage of missing values and to rank fields based on importance relative to a specified target.
- 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.

Filling in 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.
- 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.
- 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 specifying 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 that 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:

- count_nulls(LIST)
- @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 count nulls across a list of fields, as follows:

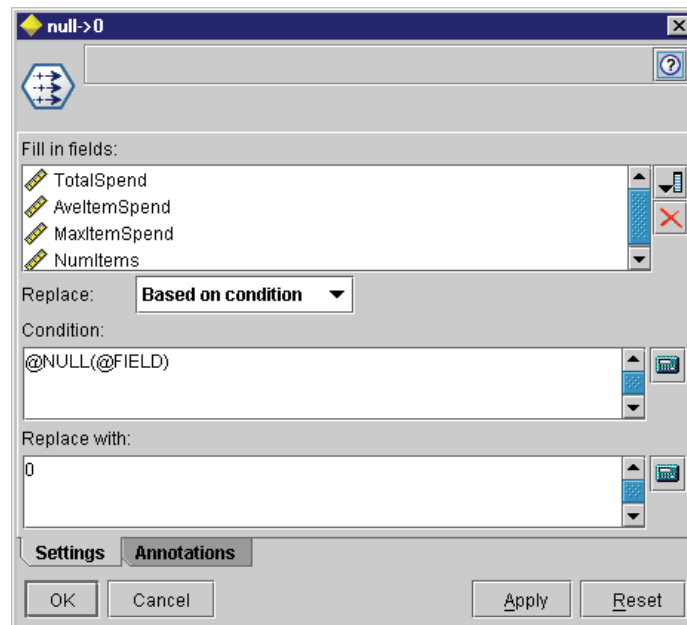
```
count_nulls(['cardtenure' 'card2tenure' 'card3tenure'])
```

When using any of the functions that accept a list of fields as input, the special functions @FIELDS_BETWEEN and @FIELDS_MATCHING can be used, as shown in the following example:

```
count_nulls(@FIELDS_MATCHING('card*'))
```

Figure 6-3

Using a Filler node to replace null values with 0 in the selected fields



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:

```
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 Chapter 8 on p. 162.

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 by 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:

```
Drug = 'drugC' and not(@NULL(Drug))
```

Earlier versions of Clementine excluded null values in such situations.

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 in either the user interface or batch mode. This allows you to perform many of the same data manipulations in an automated fashion.

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 the 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' >= -0.2635771036148072 and 'KX-Kohonen' <= 0.3146203637123107  
and 'KY-Kohonen' >= -0.18975617885589602 and  
'KY-Kohonen' <= 0.17674794197082522) -> T
```

Several functions, such as string functions, require you to enter several parameters using correct syntax. For example, the function *subscrs* is used below to return the first character of a *produce_ID* field, indicating whether an item is organic, genetically modified, or conventional. The results of an expression are described by *"-> Result"*:

```
subscrs(1,produce_ID) -> `c`
```

Similarly, the following expression is

```
stripchar(`3`,`123`) -> "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 `subscr` and `lowertoupper` to return the first character of `produce_ID` and convert it to upper case.

```
lowertoupper(subscr(1,produce_ID)) -> `C`
```

This same expression can be written in shorthand as:

```
lowertoupper(produce_ID(1)) -> `C`
```

Another commonly used combination of functions is:

```
locchar_back(`n`, (length(web_page)), web_page)
```

This expression locates the character ``n`` within the values of the field `web_page` reading backward 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 combination, which is 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 Chapter 8 on p. 129.

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-NextField'

where *Product* is the name of a field from a market basket data set, '*\$P-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 backquotes 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 U.K. and U.S. keyboard, the key for the backquote character (grave accent, Unicode 0060) can be found just below the Esc 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)
```

Stream, Session, and SuperNode Parameters

Parameters can be defined in Clementine for use in CLEM expressions and in scripting. They are, in effect, user-defined variables that are saved and persisted with the current stream, session, or SuperNode, and they can be accessed from the user interface as well as through scripting. If you save a stream, for example, any parameters set for that stream are also saved. (This distinguishes them from local script variables, which can be used only in the script where they are declared.) Parameters are often used in scripting as part of a CLEM expression in which the parameter value is specified in the script.

The scope of a parameter depends on where it is set:

- **Stream parameters** can be set in a stream script or in the stream properties dialog box, and they are available to all nodes in the stream. They will appear on the Parameters drop-down list in the Expression Builder.
- **Session parameters** can be set in a stand-alone script or in the session parameters dialog box. They are available to all streams used in a single instance of Clementine (all streams listed on the Streams tab in the managers window).

Parameters can also be set for SuperNodes, in which case they are visible only to nodes encapsulated within that SuperNode.

Using Parameters in CLEM Expressions

Parameters are represented in CLEM expressions by `$P-pname`, where `pname` is the name of the parameter. When used in CLEM expressions, parameters must be placed within single quotes—for example, `'$P-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. For more information, see “Selecting Fields, Parameters, and Global Variables” on p. 125.

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—`length(STRING)`.
- Checking the alphabetical ordering of string values—`alphabefore(STRING1, STRING2)`.

- Removing leading or trailing whitespace from values—`trim(String)`, `trim_start(String)`, or `trimend(String)`.
- Replacing all instances of a substring within a string—`replace(SUBSTRING, NEWSUBSTRING, STRING)`. For example, you could strip unsupported or non-English characters from a field and replace them with a single space prior to text mining.
- Extract the first or last n characters from a string—`startstring(LENGTH, STRING)` or `endstring(LENGTH, STRING)`. For example, if you have a field that combines a product name with a 4-digit ID code (ACME CAMERA-D109), you could extract just the code.
- Matching a specific pattern—`STRING` matches `PATTERN`. For example `job_title` matches `"*market"` would match persons with “market” anywhere in their job title.

For more information, see “String Functions” in Chapter 8 on p. 145.

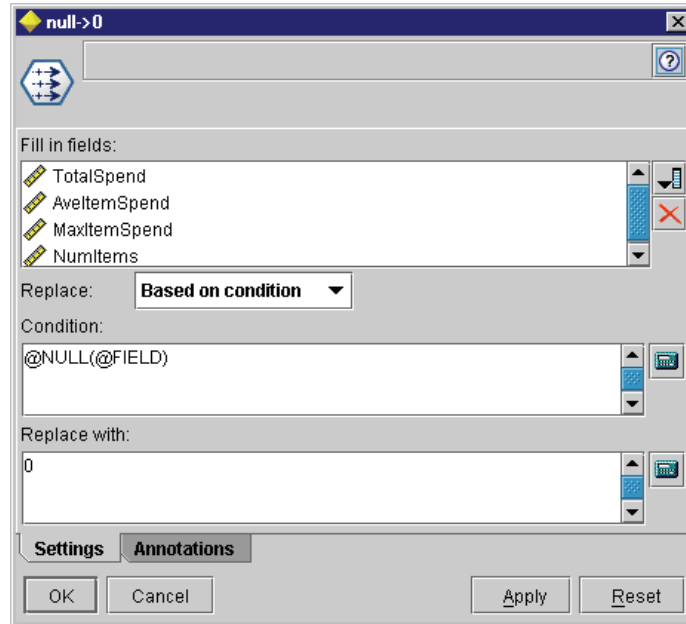
Handling Blanks and Missing Values

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:

- `@BLANK(FIELD)` can be used to determine records whose values are blank for a particular field, such as *Age*.
- `@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.

Figure 7-1

Using @NULL to fill missing values in the selected fields with 0



For more information, see “Functions Handling Blanks and Null Values” in Chapter 8 on p. 162.

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 Chapter 8 on p. 141.

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 the date string `DATE` as a real number. This is an approximate figure, based on a month of 30.0 days.

```
date_in_months(Date)
```

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 the date string `DATE1` represents a date prior to that represented by the date string `DATE2`. Otherwise, this function returns a value of 0.

```
date_before(Date1, Date2)
```

Calculating Differences

You can also calculate the difference between two times and two dates using functions, such as:

```
date_weeks_difference(Date1, Date2)
```

This function returns the time in weeks from the date represented by the date string `DATE1` to the date represented by the date string `DATE2` as a real number. This is based on a week of 7.0 days. If `DATE2` is prior to `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 Chapter 8 on p. 151.

Summarizing Multiple Fields

The CLEM language includes a number of functions that return summary statistics across multiple fields. These functions may be particularly useful in analyzing survey data, where multiple responses to a question may be stored in multiple fields.

Comparison Functions

You can compare values across multiple fields using the min_n and max_n functions; for example:

```
max_n(['card1fee' 'card2fee' 'card3fee' 'card4fee'])
```

You can also use a number of counting functions to obtain counts of values that meet specific criterion, even when those values are stored in multiple fields. For example, to count the number of cards that have been held for more than five years:

```
count_greater_than(5, ['cardtenure' 'card2tenure' 'card3tenure'])
```

To count null values across the same set of fields:

```
count_nulls(['cardtenure' 'card2tenure' 'card3tenure'])
```

Note that this example counts the number of cards being held, not the number of people holding them. For more information, see “Comparison Functions” in Chapter 8 on p. 139.

Numeric Functions

You can obtain statistics across multiple fields using the sum_n, mean_n, and sdev_n functions; for example:

```
sum_n(['card1bal' 'card2bal' 'card3bal'])
```

```
mean_n(['card1bal' 'card2bal' 'card3bal'])
```

For more information, see “Numeric Functions” in Chapter 8 on p. 141.

Generating Lists of Fields

When using any of the functions that accept a list of fields as input, the special functions `@FIELDS_BETWEEN(start, end)` and `@FIELDS_MATCHING(pattern)` can be used as input. For example, assuming the order of fields is as shown in the `sum_n` example above, the following would be equivalent:

```
sum_n(@FIELDS_BETWEEN(card1bal, card3bal))
```

Alternatively, to count the number of null values across all fields beginning with “*card*”:

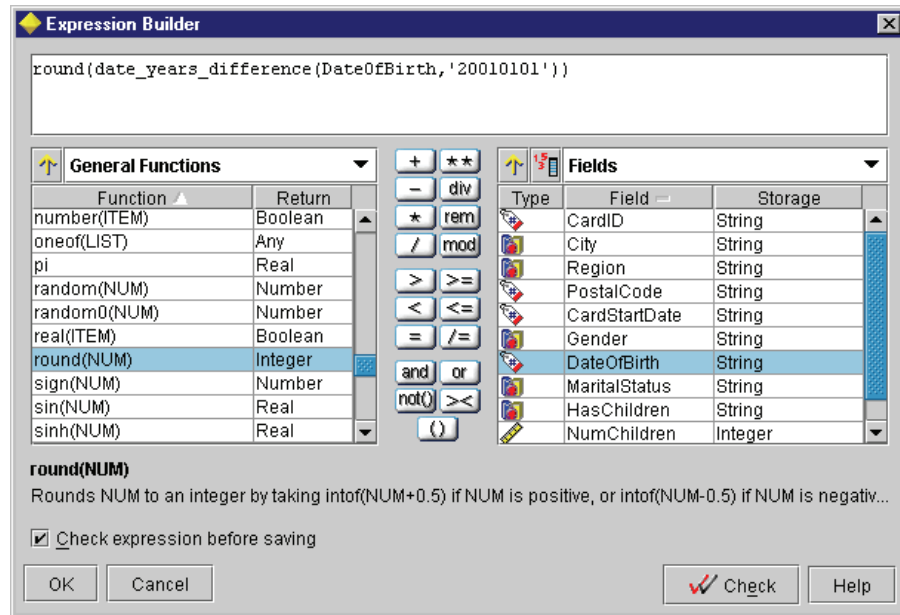
```
count_nulls(@FIELDS_MATCHING('card*'))
```

For more information, see “Special Fields” in Chapter 8 on p. 163.

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.

Figure 7-2
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.

To access the Expression Builder:

- ▶ Click the calculator button on the right side of the dialog box.

Figure 7-3

Calculator button used to access the Expression Builder***Creating Expressions***

The Expression Builder provides not only complete lists of fields, functions, and operators but also access to data values if your data is instantiated.

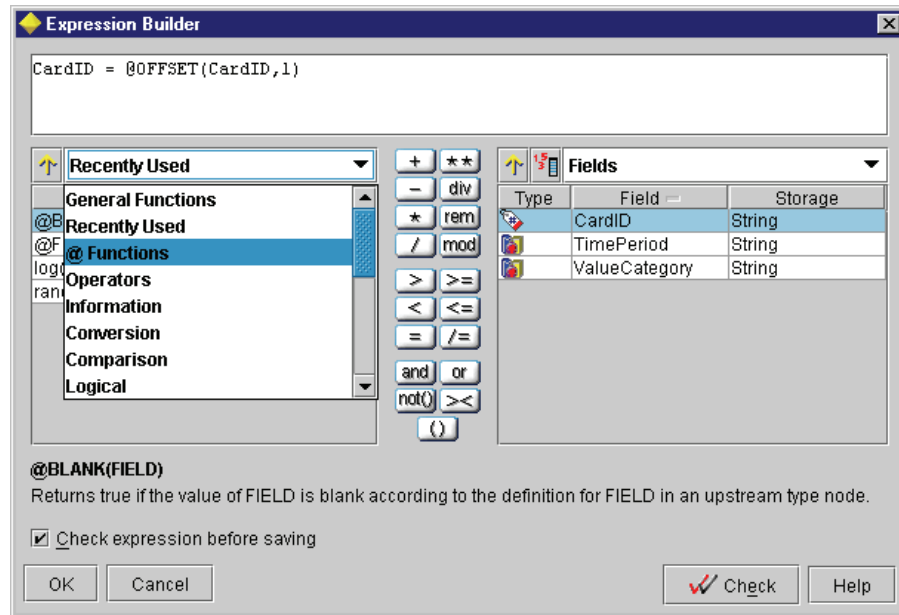
To create an expression using the Expression Builder:

- ▶ 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.

Figure 7-4
Expression Builder: Functions list



Available functions are grouped into categories for easier searching. Note that there are two categories that you may find particularly useful:

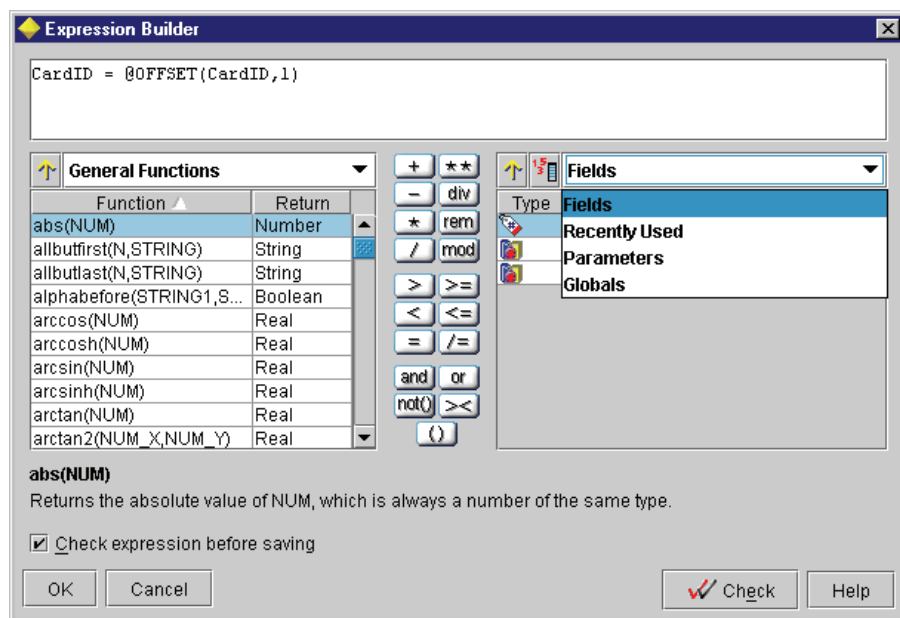
- **All Functions** contains a complete list of available CLEM functions.
- **Recently Used** contains a list of CLEM functions used within the current session.

After you have selected a group of functions, double-click to insert the functions into 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.

Figure 7-5
Expression Builder: Fields list

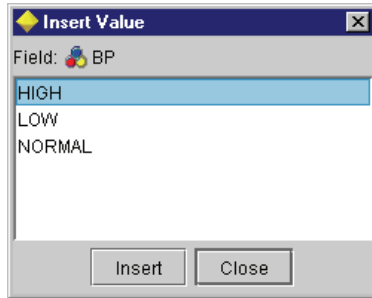


For more information, see “Stream, Session, and SuperNode Parameters” on p. 116.

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.

Figure 7-6
Field values dialog box with Expression Builder options



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.

Figure 7-7
Value picker button



- ▶ Select a value from the list.
- ▶ Click Insert to add it to the CLEM expression at the cursor position.

Checking CLEM Expressions

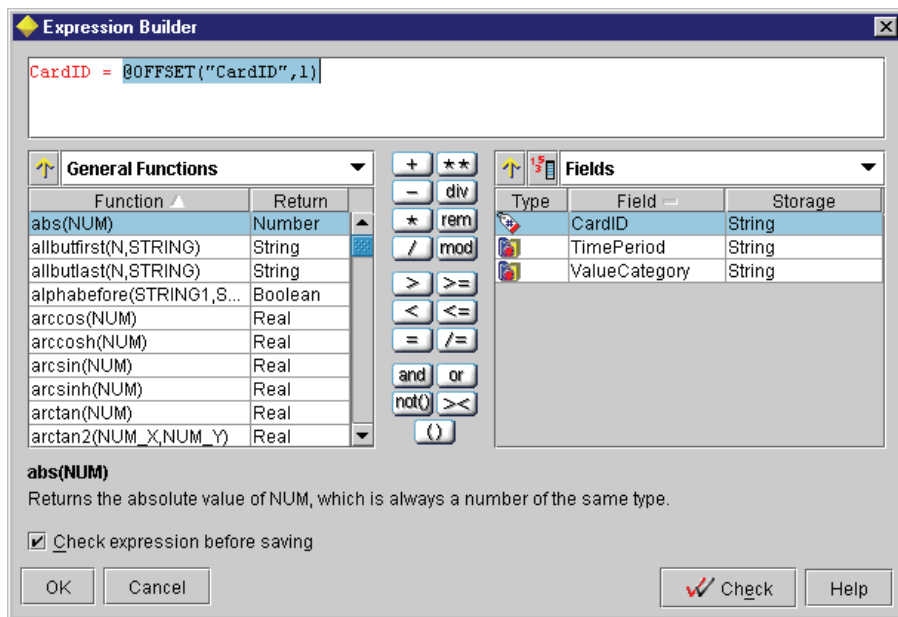
Before closing the Expression Builder, take a moment to check the function that you created. Unchecked expressions are displayed in red. Click Check to validate the expression, checking the following:

- Correct quoting of values and field names
- Correct 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.

Figure 7-8
Invalid CLEM expression



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.

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 tasks ranging from evaluating conditions or deriving values to inserting data into reports. For more information, see “What Is CLEM?” in Chapter 7 on p. 111.

A subset of the CLEM language can also be used when you are scripting in either the user interface or batch mode. This allows you to perform many of the same data manipulations in an automated fashion.

CLEM expressions consist of values, field names, operators, and functions. Using the correct syntax, you can create a wide variety of powerful data operations. For more information, see “CLEM Examples” in Chapter 7 on p. 111.

CLEM Datatypes

CLEM datatypes can be made up of any of the following:

- Integers
- Reals
- Characters
- Strings
- Lists
- Fields
- Date/Time

Rules for Quoting

Although Clementine is flexible when you are determining the fields, values, parameters, and strings used in a CLEM expression, the following general rules provide a list of “good practices” to use in creating expressions:

- **Strings**—Always use double quotes when writing strings, such as "Type 2". Single quotes can 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 7 on p. 114. 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 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 upper case. 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, `"$65443"`. You can use single quotes around a string, but the result is indistinguishable from a quoted field (`'referrerID'`). For more information, see “String Functions” on p. 145.

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-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:

Format	Examples
DDMMYY	150163
MMDDYY	011563
YYMMDD	630115
YYYYMMDD	19630115
YYYYDDD	Four-digit year, followed by a three-digit number representing the day of the year—e.g., 2000032 represents the 32nd day of 2000 or February 1, 2000
DAY	Day of the week in the current locale—e.g., Monday, Tuesday, ..., etc., in English
MONTH	Month in the current locale—e.g., January, February, ..., etc.
DD/MM/YY	15/01/63
DD/MM/YYYY	15/01/1963
MM/DD/YY	01/15/63
MM/DD/YYYY	01/15/1963
DD-MM-YY	15-01-63
DD-MM-YYYY	15-01-1963
MM-DD-YY	01-15-63
MM-DD-YYYY	01-15-1963
DD.MM.YY	15.01.63

Format	Examples
DD.MM.YYYY	15.01.1963
MM.DD.YY	01.15.63
MM.DD.YYYY	01.15.1963
DD-MON-YY	15-JAN-63, 15-jan-63, 01-Jan-63
DD/MON/YY	15/JAN/63, 15/jan/63, 01/Jan/63
DD.MON.YY	15.JAN.63, 15.jan.63, 01.Jan.63
DD-MON-YYYY	15-JAN-1963, 15-jan-1963, 01-Jan-1963
DD/MON/YYYY	15/JAN/1963, 15/jan/1963, 01/Jan/1963
DD.MON.YYYY	15.JAN.1963, 15.jan.1963, 01.Jan.1963
MON YYYY	Jan 2004
q Q YYYY	Date represented as a digit (1–4) representing the quarter followed by the letter <i>Q</i> and a four-digit year—e.g., 25th Dec 2004 would be represented as 4 Q 2004
ww WK YYYY	Two-digit number representing the week of the year, followed by the letters <i>WK</i> and then a four-digit year. The week of the year is calculated assuming that the first day of the week is Monday and there is at least one day in the first week.

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:

Format	Examples
HHMMSS	120112, 010101, 221212
HHMM	1223, 0745, 2207
MMSS	5558, 0100
HH:MM:SS	12:01:12, 01:01:01, 22:12:12
HH:MM	12:23, 07:45, 22:07
MM:SS	55:58, 01:00

Format	Examples
(H)H:(M)M:(S)S	12:1:12, 1:1:1, 22:12:12
(H)H:(M)M	12:23, 7:45, 22:7
(M)M:(S)S	55:58, 1:0
HH.MM.SS	12.01.12, 01.01.01, 22.12.12
HH.MM	12.23, 07.45, 22.07
MM.SS	55.58, 01.00
(H)H.(M)M.(S)S	12.1.12, 1.1.1, 22.12.12
(H)H.(M)M	12.23, 7.45, 22.7
(M)M.(S)S	55.58, 1.0

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:

Operation	Precedence
or	10
and	9
=	7
==	7
/=	7
/==	7
>	6
>=	6

Operation	Precedence
<	6
<=	6
&&=_0	6
&&/=_0	6
+	5
><	5
-	5
*	4
&&	4
&&~~	4
	4
~~	4
/&	4
<<	4
>>	4
/	4
**	3
rem	2
div	2

Functions Reference

The following CLEM functions are available for 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.

Function Type	Description
Information	Used to gain insight into field values. For example, the function <code>is_string</code> returns true for all records whose type is a string.
Conversion	Used to construct new fields or convert storage type. For example, the function <code>to_timestamp</code> converts the selected field to a timestamp.

Function Type	Description
Comparison	Used to compare field values to each other or to a specified string. For example, <= is used to compare whether the values of two fields are lesser or equal.
Logical	Used to perform logical operations, such as if, then, else operations.
Numeric	Used to perform numeric calculations, such as the natural log of field values.
Trigonometric	Used to perform trigonometric calculations, such as the arccosine of a specified angle.
Probability	Return probabilities based on various distributions, such as probability that a value from Student's <i>t</i> distribution will be less than a specific value.
Bitwise	Used to manipulate integers as bit patterns.
Random	Used to randomly select items or generate numbers.
String	Used to perform a wide variety of operations on strings, such as stripchar, which allows you to remove a specified character.
SoundEx	Used find strings when the precise spelling is not known, based on phonetic assumptions about how certain letters are pronounced.
Date and time	Used to perform a variety of operations on datetime fields.
Sequence	Used to gain insight into the record sequence of a data set or perform operations based on that sequence.
Global	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.
Blanks and null	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.
Special fields	Used to denote the specific fields under examination. For example, @FIELD is used when deriving multiple fields.

Conventions in Function Descriptions

The following conventions are used throughout this guide when referring to items in a function:

<i>BOOL</i>	A Boolean, or flag, such as true or false
<i>NUM, NUM1, NUM2</i>	Any number

<i>REAL, REAL1, REAL2</i>	Any real number, such as 1.234, –77.01
<i>INT, INT1, INT2</i>	Any integer, such as 1 or –77
<i>CHAR</i>	A character code, such as `A`
<i>STRING</i>	A string, such as "referrerID"
<i>LIST</i>	A list of items, such as ["abc" "def"]
<i>ITEM</i>	A field, such as Customer or extract_concept
<i>DATE</i>	A date field, such as start_date, where values are in a format such as DD-MON-YYYY
<i>TIME</i>	A time field, such as power_flux, where values are in a format such as HHMMSS

Functions in this guide are listed with the function in one column, the result type (integer, string, etc.) in another, and a description (where available) in a third column. For example, the rem function description is shown below.

Function	Result	Description
INT1 rem INT2	<i>Number</i>	Returns the remainder of <i>INT1</i> divided by <i>INT2</i> . For example, INT1 – (INT1 div INT2) * INT2.

Details on usage conventions, such as how to list items or specify characters in a function, are described elsewhere. For more information, see “CLEM Datatypes” on p. 129.

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.

Function	Result	Description
@BLANK(FIELD)	<i>Boolean</i>	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). Note that this function cannot be called from a script.

Function	Result	Description
@NULL(ITEM)	<i>Boolean</i>	Returns true for all records whose values are undefined. Undefined values are system null values, displayed in Clementine as \$null\$. Note that this function cannot be called from a script.
is_date(ITEM)	<i>Boolean</i>	Returns true for all records whose type is a date.
is_datetime(ITEM)	<i>Boolean</i>	Returns true for all records whose type is datetime.
is_integer(ITEM)	<i>Boolean</i>	Returns true for all records whose type is an integer.
is_number(ITEM)	<i>Boolean</i>	Returns true for all records whose type is a number.
is_real(ITEM)	<i>Boolean</i>	Returns true for all records whose type is a real.
is_string(ITEM)	<i>Boolean</i>	Returns true for all records whose type is a string.
is_time(ITEM)	<i>Boolean</i>	Returns true for all records whose type is time.
is_timestamp(ITEM)	<i>Boolean</i>	Returns true for all records whose type is a timestamp.

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, then V1 >< V2 returns "35" (a string, not a number).

Function	Result	Description
ITEM1 >< ITEM2	<i>String</i>	Concatenates values for two fields and returns the resulting string as <i>ITEM1ITEM2</i> .
to_integer(ITEM)	<i>Integer</i>	Converts the storage of the specified field to an integer.
to_real(ITEM)	<i>Real</i>	Converts the storage of the specified field to a real.
to_number(ITEM)	<i>Number</i>	Converts the storage of the specified field to a number.
to_string(ITEM)	<i>String</i>	Converts the storage of the specified field to a string.
to_time(ITEM)	<i>Time</i>	Converts the storage of the specified field to a time.
to_date(ITEM)	<i>Date</i>	Converts the storage of the specified field to a date.
to_timestamp(ITEM)	<i>Timestamp</i>	Converts the storage of the specified field to a timestamp.
to_datetime(ITEM)	<i>Datetime</i>	Converts the storage of the specified field to a datetime value.

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.

NUM1 = NUM2 -> BOOL

Function	Result	Description
<code>ITEM1 = ITEM2</code>	<i>Boolean</i>	Returns true for records where <i>ITEM1</i> is equal to <i>ITEM2</i> .
<code>ITEM1 /= ITEM2</code>	<i>Boolean</i>	Returns true if the two strings are not identical and 0 if they are identical.
<code>ITEM1 < ITEM2</code>	<i>Boolean</i>	Returns true for records where <i>ITEM1</i> is less than <i>ITEM2</i> .
<code>ITEM1 <= ITEM2</code>	<i>Boolean</i>	Returns true for records where <i>ITEM1</i> is less than or equal to <i>ITEM2</i> .
<code>ITEM1 > ITEM2</code>	<i>Boolean</i>	Returns true for records where <i>ITEM1</i> is greater than <i>ITEM2</i> .
<code>ITEM1 >= ITEM2</code>	<i>Boolean</i>	Returns true for records where <i>ITEM1</i> is greater than or equal to <i>ITEM2</i> .
<code>count_equal(ITEM1, LIST)</code>	<i>Integer</i>	Returns the number of values from a list of fields that are equal to <i>ITEM1</i> , or null if <i>ITEM1</i> is null. For more information, see “Summarizing Multiple Fields” in Chapter 7 on p. 121.
<code>count_greater_than(ITEM1, LIST)</code>	<i>Integer</i>	Returns the number of values from a list of fields that are greater than <i>ITEM1</i> , or null if <i>ITEM1</i> is null.
<code>count_less_than(ITEM1, LIST)</code>	<i>Integer</i>	Returns the number of values from a list of fields that are less than <i>ITEM1</i> , or null if <i>ITEM1</i> is null.
<code>count_not_equal(ITEM1, LIST)</code>	<i>Integer</i>	Returns the number of values from a list of fields that are not equal to <i>ITEM1</i> , or null if <i>ITEM1</i> is null.
<code>count_nulls(LIST)</code>	<i>Integer</i>	Returns the number of null values from a list of fields.
<code>date_before(DATE1, DATE2)</code>	<i>Boolean</i>	Used to check the ordering of date values. Returns a true value if <i>DATE1</i> is before <i>DATE2</i> .
<code>max(ITEM1, ITEM2)</code>	<i>Any</i>	Returns the greater of the two items— <i>ITEM1</i> or <i>ITEM2</i> .
<code>max_n(LIST)</code>	<i>Number</i>	Returns the maximum value from a list of numeric fields or null if all of the field values are null. For more information, see “Summarizing Multiple Fields” in Chapter 7 on p. 121.
<code>min(ITEM1, ITEM2)</code>	<i>Any</i>	Returns the lesser of the two items— <i>ITEM1</i> or <i>ITEM2</i> .

Function	Result	Description
min_n(LIST)	<i>Number</i>	Returns the minimum value from a list of numeric fields or null if all of the field values are null.
time_before(TIME1, TIME2)	<i>Boolean</i>	Used to check the ordering of time values. Returns a true value if <i>TIME1</i> is before <i>TIME2</i> .
member(ITEM, LIST)	<i>Boolean</i>	Returns true if <i>ITEM</i> is a member of the specified <i>LIST</i> . Otherwise, a false value is returned. A list of field names can also be specified. For more information, see “Summarizing Multiple Fields” in Chapter 7 on p. 121.

Logical Functions

CLEM expressions can be used to perform logical operations.

Function	Result	Description
COND1 and COND2	<i>Boolean</i>	This operation is a logical conjunction and returns a true value if both <i>COND1</i> and <i>COND2</i> are true. If <i>COND1</i> is false, then <i>COND2</i> is not evaluated; this makes it possible to have conjunctions where <i>COND1</i> first tests that an operation in <i>COND2</i> is legal. For example, length(Label) >=6 and Label(6) = 'x'.
COND1 or COND2	<i>Boolean</i>	This operation is a logical (inclusive) disjunction and returns a true value if either <i>COND1</i> or <i>COND2</i> is true or if both are true. If <i>COND1</i> is true, <i>COND2</i> is not evaluated.
not(COND)	<i>Boolean</i>	This operation is a logical negation and returns a true value if <i>COND</i> is false. Otherwise, this operation returns a value of 0.
if COND then EXPR1 else EXPR2 endif	<i>Any</i>	This operation is a conditional evaluation. If <i>COND</i> is true, this operation returns the result of <i>EXPR1</i> . Otherwise, the result of evaluating <i>EXPR2</i> is returned.
if COND1 then EXPR1 elseif COND2 then EXPR2 else EXPR_N endif	<i>Any</i>	This operation is a multibranch conditional evaluation. If <i>COND1</i> is true, this operation returns the result of <i>EXPR1</i> . Otherwise, if <i>COND2</i> is true, this operation returns the result of evaluating <i>EXPR2</i> . Otherwise, the result of evaluating <i>EXPR_N</i> is returned.

Numeric Functions

CLEM contains a number of commonly used numeric functions.

Function	Result	Description
<code>-NUM</code>	<i>Number</i>	Used to negate <i>NUM</i> . Returns the corresponding number with the opposite sign.
<code>NUM1 + NUM2</code>	<i>Number</i>	Returns the sum of <i>NUM1</i> and <i>NUM2</i> .
<code>code -NUM2</code>	<i>Number</i>	Returns the value of <i>NUM2</i> subtracted from <i>NUM1</i> .
<code>NUM1 * NUM2</code>	<i>Number</i>	Returns the value of <i>NUM1</i> multiplied by <i>NUM2</i> .
<code>NUM1 / NUM2</code>	<i>Number</i>	Returns the value of <i>NUM1</i> divided by <i>NUM2</i> .
<code>INT1 div INT2</code>	<i>Number</i>	Used to perform integer division. Returns the value of <i>INT1</i> divided by <i>INT2</i> .
<code>INT1 rem INT2</code>	<i>Number</i>	Returns the remainder of <i>INT1</i> divided by <i>INT2</i> . For example, <i>INT1 - (INT1 div INT2) * INT2</i> .
<code>INT1 mod INT2</code>	<i>Number</i>	This function has been deprecated in favor of the <code>rem</code> function and should no longer be used.
<code>BASE ** POWER</code>	<i>Number</i>	Returns <i>BASE</i> raised to the power <i>POWER</i> , where either may be any number (except that <i>BASE</i> must not be zero if <i>POWER</i> is zero of any type other than integer 0). If <i>POWER</i> is an integer, the computation is performed by successively multiplying powers of <i>BASE</i> . Thus, if <i>BASE</i> is an integer, the result will be an integer. If <i>POWER</i> is integer 0, the result is always a 1 of the same type as <i>BASE</i> . Otherwise, if <i>POWER</i> is not an integer, the result is computed as <code>exp(POWER * log(BASE))</code> .
<code>abs(NUM)</code>	<i>Number</i>	Returns the absolute value of <i>NUM</i> , which is always a number of the same type.
<code>exp(NUM)</code>	<i>Real</i>	Returns <i>e</i> raised to the power <i>NUM</i> , where <i>e</i> is the base of natural logarithms.
<code>fracof(NUM)</code>	<i>Real</i>	Returns the fractional part of <i>NUM</i> , defined as <code>NUM-intof(NUM)</code> .
<code>intof(NUM)</code>	<i>Integer</i>	Truncates its argument to an integer. It returns the integer of the same sign as <i>NUM</i> and with the largest magnitude such that <code>abs(INT) <= abs(NUM)</code> .
<code>log(NUM)</code>	<i>Real</i>	Returns the natural (base <i>e</i>) logarithm of <i>NUM</i> , which must not be a zero of any kind.
<code>log10(NUM)</code>	<i>Real</i>	Returns the base 10 logarithm of <i>NUM</i> , which must not be a zero of any kind. This function is defined as <code>log(NUM) / log(10)</code> .
<code>negate(NUM)</code>	<i>Number</i>	Used to negate <i>NUM</i> . Returns the corresponding number with the opposite sign.

Function	Result	Description
round(NUM)	<i>Integer</i>	Used to round <i>NUM</i> to an integer by taking <code>intof(NUM+0.5)</code> if <i>NUM</i> is positive or <code>intof(NUM-0.5)</code> if <i>NUM</i> is negative.
sign(NUM)	<i>Number</i>	Used to determine the sign of <i>NUM</i> , this operation returns -1, 0, or 1 if <i>NUM</i> is an integer. If <i>NUM</i> is a real, it returns -1.0, 0.0, or 1.0, depending on whether <i>NUM</i> is negative, zero, or positive.
sqrt(NUM)	<i>Real</i>	Returns the square root of <i>NUM</i> . <i>NUM</i> must be positive.
sum_n(LIST)	<i>Number</i>	Returns the sum of values from a list of numeric fields or null if all of the field values are null. For more information, see “Summarizing Multiple Fields” in Chapter 7 on p. 121.
mean_n(LIST)	<i>Number</i>	Returns the mean value from a list of numeric fields or null if all of the field values are null.
sdev_n(LIST)	<i>Number</i>	Returns the standard deviation from a list of numeric fields or null if all of the field values are null.

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.

Function	Result	Description
arccos(NUM)	<i>Real</i>	Computes the arccosine of the specified angle.
arccosh(NUM)	<i>Real</i>	Computes the hyperbolic arccosine of the specified angle.
arcsin(NUM)	<i>Real</i>	Computes the arcsine of the specified angle.
arcsinh(NUM)	<i>Real</i>	Computes the hyperbolic arcsine of the specified angle.
arctan(NUM)	<i>Real</i>	Computes the arctangent of the specified angle.
arctan2(NUM_X, NUM_Y)	<i>Real</i>	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 $-\pi < \text{ANGLE} \leq \pi$ (radians) – $180 < \text{ANGLE} \leq 180$ (degrees)
arctanh(NUM)	<i>Real</i>	Computes the hyperbolic arctangent of the specified angle.
cos(NUM)	<i>Real</i>	Computes the cosine of the specified angle.
cosh(NUM)	<i>Real</i>	Computes the hyperbolic cosine of the specified angle.
pi	<i>Real</i>	This constant is the best real approximation to π .
sin(NUM)	<i>Real</i>	Computes the sine of the specified angle.
sinh(NUM)	<i>Real</i>	Computes the hyperbolic sine of the specified angle.

Function	Result	Description
tan(NUM)	<i>Real</i>	Computes the tangent of the specified angle.
tanh(NUM)	<i>Real</i>	Computes the hyperbolic tangent of the specified angle.

Probability Functions

Probability functions return probabilities based on various distributions, such as the probability that a value from Student's *t* distribution will be less than a specific value.

Function	Result	Description
cdf_chisq(NUM, DF)	<i>Real</i>	Returns the probability that a value from the chi-square distribution with the specified degrees of freedom will be less than the specified number.
cdf_f(NUM, DF1, DF2)	<i>Real</i>	Returns the probability that a value from the <i>F</i> distribution, with degrees of freedom DF1 and DF2, will be less than the specified number.
cdf_normal(NUM, MEAN, STDDEV)	<i>Real</i>	Returns the probability that a value from the normal distribution with the specified mean and standard deviation will be less than the specified number.
cdf_t(NUM, DF)	<i>Real</i>	Returns the probability that a value from Student's <i>t</i> distribution with the specified degrees of freedom will be less than the specified number.

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.

Note: Bitwise functions cannot be called from scripts.

Function	Result	Description
~~ INT1	<i>Integer</i>	Produces the bitwise complement of the integer <i>INT1</i> . That is, there is a 1 in the result for each bit position for which <i>INT1</i> has 0. It is always true that $~~ \text{INT} = -(\text{INT} + 1)$. Note that this function cannot be called from a script.

Function	Result	Description
<code>INT1 INT2</code>	<i>Integer</i>	The result of this operation is the bitwise “inclusive or” of <i>INT1</i> and <i>INT2</i> . That is, there is a 1 in the result for each bit position for which there is a 1 in either <i>INT1</i> or <i>INT2</i> or both.
<code>INT1 & INT2</code>	<i>Integer</i>	The result of this operation is the bitwise “exclusive or” of <i>INT1</i> and <i>INT2</i> . That is, there is a 1 in the result for each bit position for which there is a 1 in either <i>INT1</i> or <i>INT2</i> but not in both.
<code>INT1 && INT2</code>	<i>Integer</i>	Produces the bitwise “and” of the integers <i>INT1</i> and <i>INT2</i> . That is, there is a 1 in the result for each bit position for which there is a 1 in both <i>INT1</i> and <i>INT2</i> .
<code>INT1 &&~ INT2</code>	<i>Integer</i>	Produces the bitwise “and” of <i>INT1</i> and the bitwise complement of <i>INT2</i> . That is, there is a 1 in the result for each bit position for which there is a 1 in <i>INT1</i> and a 0 in <i>INT2</i> . This is the same as <code>INT1 && (~INT2)</code> and is useful for clearing bits of <i>INT1</i> set in <i>INT2</i> .
<code>INT << N</code>	<i>Integer</i>	Produces the bit pattern of <i>INT1</i> shifted left by <i>N</i> positions. A negative value for <i>N</i> produces a right shift.
<code>INT >> N</code>	<i>Integer</i>	Produces the bit pattern of <i>INT1</i> shifted right by <i>N</i> positions. A negative value for <i>N</i> produces a left shift.
<code>INT1 &&=_0 INT2</code>	<i>Boolean</i>	Equivalent to the Boolean expression <code>INT1 && INT2 != 0</code> but is more efficient.
<code>INT1 &&/= _0 INT2</code>	<i>Boolean</i>	Equivalent to the Boolean expression <code>INT1 && INT2 == 0</code> but is more efficient.
<code>integer_bitcount(INT)</code>	<i>Integer</i>	Counts the number of 1 or 0 bits in the two’s-complement representation of <i>INT</i> . If <i>INT</i> is non-negative, <i>N</i> is the number of 1 bits. If <i>INT</i> 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 <code>integer_bitcount(INT) = integer_bitcount(-(INT+1))</code> .
<code>integer_leastbit(INT)</code>	<i>Integer</i>	Returns the bit position <i>N</i> of the least-significant bit set in the integer <i>INT</i> . <i>N</i> is the highest power of 2 by which <i>INT</i> divides exactly.
<code>integer_length(INT)</code>	<i>Integer</i>	Returns the length in bits of <i>INT</i> as a two’s-complement integer. That is, <i>N</i> is the smallest integer such that <code>INT < (1 << N)</code> if <code>INT >= 0</code> <code>INT >= (-1 << N)</code> if <code>INT < 0</code> . If <i>INT</i> is non-negative, then the representation of <i>INT</i> as an unsigned integer requires a field of at least <i>N</i> bit. Alternatively, a minimum of <i>N+1</i> bits is required to represent <i>INT</i> as a signed integer, regardless of its sign.
<code>testbit(INT, N)</code>	<i>Boolean</i>	Tests the bit at position <i>N</i> in the integer <i>INT</i> and returns the state of bit <i>N</i> as a Boolean value, which is true for 1 and false for 0.

Random Functions

The following functions are used to randomly select items or randomly generate numbers.

Function	Result	Description
oneof(LIST)	<i>Any</i>	Returns a randomly chosen element of <i>LIST</i> . List items should be entered as [ITEM1,ITEM2,...,ITEM_N]. Note that a list of field names can also be specified. For more information, see “Summarizing Multiple Fields” in Chapter 7 on p. 121.
random(NUM)	<i>Number</i>	Returns a uniformly distributed random number of the same type (<i>INT</i> or <i>REAL</i>), starting from 1 to <i>NUM</i> . 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 <i>NUM</i> .
random0(NUM)	<i>Number</i>	This has the same properties as <i>random(NUM)</i> , but starting from 0. The largest random number returned by the function will never equal <i>X</i> .

String Functions

In CLEM, you can perform the following operations with strings:

- Compare strings.
- Create strings.
- Access characters.

In a CLEM, 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 `z`, `A`, or `2`. Characters that are out of bounds or negative indices to a string will result in a null value.

Function	Result	Description
allbutfirst(N, STRING)	<i>String</i>	Returns a string, which is <i>STRING</i> with the first <i>N</i> characters removed.
allbutlast(N, STRING)	<i>String</i>	Returns a string, which is <i>STRING</i> with the last characters removed.

Function	Result	Description
alphabefore(String1, String2)	<i>Boolean</i>	Used to check the alphabetical ordering of strings. Returns true if <i>String1</i> precedes <i>String2</i> .
endstring(Length, String)	<i>String</i>	Extracts the last <i>n</i> characters from the specified string. If the string length is less than or equal to the specified length, then it is unchanged.
hasendstring(String, Substring)	<i>Integer</i>	This function is the same as <code>isendstring(Sub_String, String)</code> .
hasmidstring(String, Substring)	<i>Integer</i>	This function is the same as <code>ismidstring(Sub_String, String)</code> (embedded substring).
hasstartstring(String, Substring)	<i>Integer</i>	This function is the same as <code>isstartstring(Sub_String, String)</code> .
hassubstring(String, N, Substring)	<i>Integer</i>	This function is the same as <code>issubstring(Sub_String, N, String)</code> , where <i>N</i> defaults to 1.
count_substring(String, Substring)	<i>Integer</i>	Returns the number of times the specified substring occurs within string. For example, <code>count_substring("foooo.txt", "oo")</code> returns 3.
hassubstring(String, Substring)	<i>Integer</i>	This function is the same as <code>issubstring(Sub_String, 1, String)</code> , where <i>N</i> defaults to 1.
isalphacode(Char)	<i>Boolean</i>	Returns a value of true if <i>Char</i> 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, <code>isalphacode(produce_num(1))</code> .
isendstring(Substring, String)	<i>Integer</i>	If the string <i>String</i> ends with the substring <i>Sub_String</i> , then this function returns the integer subscript of <i>Sub_String</i> in <i>String</i> . Otherwise, this function returns a value of 0.
islowercode(Char)	<i>Boolean</i>	Returns a value of true if <i>Char</i> is a lowercase letter character for the specified string (often a field name). Otherwise, this function returns a value of 0. For example, both <code>islowercode('') → T</code> and <code>islowercode(country_name(2)) → T</code> are valid expressions.
ismidstring(Substring, String)	<i>Integer</i>	If <i>Sub_String</i> is a substring of <i>String</i> but does not start on the first character of <i>String</i> or end on the last, then this function returns the subscript at which the substring starts. Otherwise, this function returns a value of 0.

Function	Result	Description
isnumbercode(CHAR)	<i>Boolean</i>	Returns a value of true if <i>CHAR</i> 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, <code>isnumbercode(product_id(2))</code> .
isstartstring(SUBSTRING, STRING)	<i>Integer</i>	If the string <i>STRING</i> starts with the substring <i>SUB_STRING</i> , then this function returns the subscript 1. Otherwise, this function returns a value of 0.
issubstring(SUBSTRING, N, STRING)	<i>Integer</i>	Searches the string <i>STRING</i> , starting from its <i>N</i> th character, for a substring equal to the string <i>SUB_STRING</i> . If found, this function returns the integer subscript at which the matching substring begins. Otherwise, this function returns a value of 0. If <i>N</i> is not given, this function defaults to 1.
issubstring(SUBSTRING, STRING)	<i>Integer</i>	Searches the string <i>STRING</i> , starting from its <i>N</i> th character, for a substring equal to the string <i>SUB_STRING</i> . If found, this function returns the integer subscript at which the matching substring begins. Otherwise, this function returns a value of 0. If <i>N</i> is not given, this function defaults to 1.
issubstring_count(SUBSTRING, N, STRING):	<i>Integer</i>	Returns the index of the <i>N</i> th occurrence of <i>SUBSTRING</i> within the specified <i>STRING</i> . If there are fewer than <i>N</i> occurrences of <i>SUBSTRING</i> , 0 is returned.
issubstring_lim(SUBSTRING, N, STARTLIM, ENDLIM, STRING)	<i>Integer</i>	This function is the same as <code>issubstring</code> , but the match is constrained to start on or before the subscript <i>STARTLIM</i> and to end on or before the subscript <i>ENDLIM</i> . The <i>STARTLIM</i> or <i>ENDLIM</i> constraints may be disabled by supplying a value of false for either argument—for example, <code>issubstring_lim(SUB_STRING, N, false, false, STRING)</code> is the same as <code>issubstring</code> .
isuppercode(CHAR)	<i>Boolean</i>	Returns a value of true if <i>CHAR</i> is an uppercase letter character. Otherwise, this function returns a value of 0. For example, both <code>isuppercode('')</code> → T and <code>isuppercode(country_name(2))</code> → T are valid expressions.
last(CHAR)	<i>String</i>	Returns the last character <i>CHAR</i> of <i>STRING</i> (which must be at least one character long).

Function	Result	Description
length(<i>STRING</i>)	<i>Integer</i>	Returns the length of the string <i>STRING</i> —that is, the number of characters in it.
locchar(<i>CHAR</i> , <i>N</i> , <i>STRING</i>)	<i>Integer</i>	Used to identify the location of characters in symbolic fields. The function searches the string <i>STRING</i> for the character <i>CHAR</i> , starting the search at the <i>N</i> th character of <i>STRING</i> . This function returns a value indicating the location (starting at <i>N</i>) 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 (<i>N</i>) (for example, an offset that is beyond the length of the string), this function returns \$null\$. For example, locchar(`n`, 2, web_page) searches the field called <i>web_page</i> for the `n` character beginning at the second character in the field value. <i>Note:</i> Be sure to use single backquotes to encapsulate the specified character.
locchar_back(<i>CHAR</i> , <i>N</i> , <i>STRING</i>)	<i>Integer</i>	Similar to locchar, except that the search is performed backward , starting from the <i>N</i> th character. For example, locchar_back(`n`, 9, web_page) searches the field <i>web_page</i> starting from the ninth character and moving backward toward 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(<field>) to dynamically use the length of the current value of the field. For example, locchar_back(`n`, (length(web_page)), web_page).
matches	<i>Boolean</i>	Returns true if a string matches a specified pattern—for example, company_name matches "SPSS". The pattern must be a string literal; it cannot be a field name containing a pattern. A question mark (?) can be included in the pattern to match exactly one character; an asterisk (*) matches zero or more characters. To match a literal question mark or asterisk (rather than using these as wildcards), a backslash (\) can be used as an escape character.

Function	Result	Description
replace(SUBSTRING, NEWSUBSTRING, STRING)	<i>String</i>	Within the specified <i>STRING</i> , replace all instances of <i>SUBSTRING</i> with <i>NEWSUBSTRING</i> .
replicate(COUNT, STRING)	<i>String</i>	Returns a string that consists of the original string copied the specified number of times.
stripchar(Char,STRING)	<i>String</i>	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 <code>stripchar('\$', 'Cost')</code> returns a new field with the dollar sign removed from all values. <i>Note:</i> Be sure to use single backquotes to encapsulate the specified character.
skipchar(Char, N, STRING)	<i>Integer</i>	Searches the string <i>STRING</i> for any character other than <i>CHAR</i> , starting at the <i>N</i> th character. This function returns an integer substring indicating the point at which one is found or 0 if every character from the <i>N</i> th onward is a <i>CHAR</i> . If the function has an invalid offset (for example, an offset that is beyond the length of the string), this function returns \$null\$. <code>locchar</code> is often used in conjunction with the <code>skipchar</code> functions to determine the value of <i>N</i> (the point at which to start searching the string). For example, <code>skipchar('s', (locchar('s', 1, "MyString")), "MyString")</code> .
skipchar_back(Char, N, STRING)	<i>Integer</i>	Similar to <code>skipchar</code> , except that the search is performed backward , starting from the <i>N</i> th character.
startstring(LENGTH, STRING)	<i>String</i>	Extracts the first <i>n</i> characters from the specified string. If the string length is less than or equal to the specified length, then it is unchanged.
strmember(Char, STRING)	<i>Integer</i>	Equivalent to <code>locchar(Char, 1, STRING)</code> . It returns an integer substring indicating the point at which <i>CHAR</i> 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\$.

Function	Result	Description
subscrs(N, STRING)	<i>CHAR</i>	Returns the <i>N</i> th character <i>CHAR</i> of the input string <i>STRING</i> . This function can also be written in a shorthand form— <i>STRING</i> (<i>N</i>) -> <i>CHAR</i> . For example, <i>lowertoupper("name"(1))</i> is a valid expression.
substring(N, LEN, STRING)	<i>String</i>	Returns a string <i>SUB_STRING</i> , which consists of the <i>LEN</i> characters of the string <i>STRING</i> , starting from the character at subscript <i>N</i> .
substring_between(N1, N2, STRING)	<i>String</i>	Returns the substring of <i>STRING</i> , which begins at subscript <i>N1</i> and ends at subscript <i>N2</i> .
trim(STRING)	<i>String</i>	Removes leading and trailing whitespace characters from the specified string.
trim_start(STRING)	<i>String</i>	Removes leading whitespace characters from the specified string.
trimend(STRING)	<i>String</i>	Removes trailing whitespace characters from the specified string.
uppertolower(CHAR)uppertolower (STRING)	<i>CHAR</i> or <i>String</i>	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. <i>Note:</i> Remember to specify strings with double quotes and characters with single backquotes. Simple field names should appear without quotes.
lowertoupper(CHAR)lowertoupper (STRING)	<i>CHAR</i> or <i>String</i>	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, <i>lowertoupper(`a`)</i> , <i>lowertoupper("My string")</i> , and <i>lowertoupper(field_name(2))</i> are all valid expressions.

SoundEx Functions

SoundEx is a method used find strings when the sound is known but the precise spelling is not. Developed in 1918, it searches out words with similar sounds based on phonetic assumptions about how certain letters are pronounced. It can be used to search names in a database, for example, where spellings and pronunciations for similar names may

vary. The basic SoundEx algorithm is documented in a number of sources and, despite known limitations (for example, leading letter combinations such as ph and f will not match even though they sound the same), is supported in some form by most databases.

Function	Result	Description
soundex(<i>STRING</i>)	<i>Integer</i>	Returns the four-character SoundEx code for the specified <i>STRING</i> .
soundex_difference(<i>STRING1</i> , <i>STRING2</i>)	<i>Integer</i>	Returns an integer between 0 and 4 that indicates the number of characters that are the same in the SoundEx encoding for the two strings, where 0 indicates no similarity and 4 indicates strong similarity or identical strings.

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.

Note: Date and time functions cannot be called from scripts.

Function	Result	Description
@TODAY	<i>String</i>	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. Note that this function cannot be called from a script.
date_before(<i>DATE1</i> , <i>DATE2</i>)	<i>Boolean</i>	Returns a value of true if <i>DATE1</i> represents a date before that represented by <i>DATE2</i> . Otherwise, this function returns a value of 0.
date_days_difference(<i>DATE1</i> , <i>DATE2</i>)	<i>Integer</i>	Returns the time in days from the date represented by <i>DATE1</i> to the date represented by <i>DATE2</i> , as an integer. If <i>DATE2</i> is before <i>DATE1</i> , this function returns a negative number.

Function	Result	Description
date_in_days(<i>DATE</i>)	<i>Integer</i>	Returns the time in days from the baseline date to the date represented by <i>DATE</i> , as an integer. If <i>DATE</i> 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.
date_in_months(<i>DATE</i>)	<i>Real</i>	Returns the time in months from the baseline date to the date represented by <i>DATE</i> , as a real number. This is an approximate figure, based on a month of 30.0 days. If <i>DATE</i> 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.
date_in_weeks(<i>DATE</i>)	<i>Real</i>	Returns the time in weeks from the baseline date to the date represented by <i>DATE</i> , as a real number. This is based on a week of 7.0 days. If <i>DATE</i> 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.
date_in_years(<i>DATE</i>)	<i>Real</i>	Returns the time in years from the baseline date to the date represented by <i>DATE</i> , as a real number. This is an approximate figure based on a year of 365.0 days. If <i>DATE</i> 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.
date_months_difference (<i>DATE1</i> , <i>DATE2</i>)	<i>Real</i>	Returns the time in months from <i>DATE1</i> to <i>DATE2</i> , as a real number. This is an approximate figure based on a month of 30.0 days. If <i>DATE2</i> is before <i>DATE1</i> , this function returns a negative number.
datetime_date(<i>YEAR</i> , <i>MONTH</i> , <i>DAY</i>)	<i>Date</i>	Creates a date value for the given <i>YEAR</i> , <i>MONTH</i> , and <i>DAY</i> . The arguments must be integers.

Function	Result	Description
<code>datetime_date(ITEM)</code>	<i>Date</i>	Returns the date value for the given <i>ITEM</i> , 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 date format 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.
<code>datetime_day(DATE)</code>	<i>Integer</i>	Returns the day of the month from a given <i>DATE</i> or timestamp. The result is an integer in the range 1 to 31.
<code>datetime_day_name(DAY)</code>	<i>String</i>	Returns the full name of the given <i>DAY</i> . The argument must be an integer in the range 1 (Sunday) to 7 (Saturday).
<code>datetime_hour(TIME)</code>	<i>Integer</i>	Returns the hour from a <i>TIME</i> or timestamp. The result is an integer in the range 0 to 23.
<code>datetime_in_seconds(DATETIME)</code>	<i>Real</i>	Returns the number of seconds in a <i>DATETIME</i> .
<code>datetime_minute(TIME)</code>	<i>Integer</i>	Returns the minute from a <i>TIME</i> or timestamp. The result is an integer in the range 0 to 59.
<code>datetime_month(DATE)</code>	<i>Integer</i>	Returns the month from a <i>DATE</i> or timestamp. The result is an integer in the range 1 to 12.
<code>datetime_month_name(MONTH)</code>	<i>String</i>	Returns the full name of the given <i>MONTH</i> . The argument must be an integer in the range 1 to 12.
<code>datetime_now</code>	<i>Timestamp</i>	Returns the current time as a timestamp.
<code>datetime_second(TIME)</code>	<i>Integer</i>	Returns the second from a <i>TIME</i> or timestamp. The result is an integer in the range 0 to 59.
<code>datetime_day_short_name(DAY)</code>	<i>String</i>	Returns the abbreviated name of the given <i>DAY</i> . The argument must be an integer in the range 1 (Sunday) to 7 (Saturday).
<code>datetime_month_short_name(MONTH)</code>	<i>String</i>	Returns the abbreviated name of the given <i>MONTH</i> . The argument must be an integer in the range 1 to 12.
<code>datetime_time(HOUR, MINUTE, SECOND)</code>	<i>Time</i>	Returns the time value for the specified <i>HOUR</i> , <i>MINUTE</i> , and <i>SECOND</i> . The arguments must be integers.
<code>datetime_time(ITEM)</code>	<i>Time</i>	Returns the time value of the given <i>ITEM</i> .
<code>datetime_timestamp(YEAR, MONTH, DAY, HOUR, MINUTE, SECOND)</code>	<i>Timestamp</i>	Returns the timestamp value for the given <i>YEAR</i> , <i>MONTH</i> , <i>DAY</i> , <i>HOUR</i> , <i>MINUTE</i> , and <i>SECOND</i> .

Function	Result	Description
<code>datetime_timestamp(</code> <i>DATE</i> <code>,</code> <i>TIME</i> <code>)</code>	<i>Timestamp</i>	Returns the timestamp value for the given <i>DATE</i> and <i>TIME</i> .
<code>datetime_timestamp (</code> <i>NUMBER</i> <code>)</code>	<i>Timestamp</i>	Returns the timestamp value of the given number of seconds.
<code>datetime_weekday(</code> <i>DATE</i> <code>)</code>	<i>Integer</i>	Returns the day of the week from the given <i>DATE</i> or timestamp.
<code>datetime_year(</code> <i>DATE</i> <code>)</code>	<i>Integer</i>	Returns the year from a <i>DATE</i> or timestamp. The result is an integer such as 2002.
<code>date_weeks_difference (</code> <i>DATE1</i> <code>,</code> <i>DATE2</i> <code>)</code>	<i>Real</i>	Returns the time in weeks from the date represented by <i>DATE1</i> to the date represented by <i>DATE2</i> , as a real number. This is based on a week of 7.0 days. If <i>DATE2</i> is before <i>DATE1</i> , this function returns a negative number.
<code>date_years_difference (</code> <i>DATE1</i> <code>,</code> <i>DATE2</i> <code>)</code>	<i>Real</i>	Returns the time in years from the date represented by <i>DATE1</i> to the date represented by <i>DATE2</i> , as a real number. This is an approximate figure based on a year of 365.0 days. If <i>DATE2</i> is before <i>DATE1</i> , this function returns a negative number.
<code>time_before(</code> <i>TIME1</i> <code>,</code> <i>TIME2</i> <code>)</code>	<i>Boolean</i>	Returns a value of true if <i>TIME1</i> represents a time before that represented by <i>TIME2</i> . Otherwise, this function returns a value of 0.
<code>time_hours_difference (</code> <i>TIME1</i> <code>,</code> <i>TIME2</i> <code>)</code>	<i>Real</i>	Returns the time difference in hours between the times represented by <i>TIME1</i> and <i>TIME2</i> , as a real number. If you select Rollover days/mins in the stream properties dialog box, a higher value of <i>TIME1</i> is taken to refer to the previous day. If you do not select the rollover option, a higher value of <i>TIME1</i> causes the returned value to be negative.
<code>time_in_hours(</code> <i>TIME</i> <code>)</code>	<i>Real</i>	Returns the time in hours represented by <i>TIME</i> , as a real number. For example, under time format HHMM, the expression <code>time_in_hours('0130')</code> evaluates to 1.5.
<code>time_in_mins(</code> <i>TIME</i> <code>)</code>	<i>Real</i>	Returns the time in minutes represented by <i>TIME</i> , as a real number.
<code>time_in_secs(</code> <i>TIME</i> <code>)</code>	<i>Integer</i>	Returns the time in seconds represented by <i>TIME</i> , as an integer.

Function	Result	Description
<code>time_mins_difference(TIME1, TIME2)</code>	<i>Real</i>	Returns the time difference in minutes between the times represented by <i>TIME1</i> and <i>TIME2</i> , as a real number. If you select Rollover days/mins in the stream properties dialog box, a higher value of <i>TIME1</i> 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 <i>TIME1</i> will cause the returned value to be negative.
<code>time_secs_difference(TIME1, TIME2)</code>	<i>Integer</i>	Returns the time difference in seconds between the times represented by <i>TIME1</i> and <i>TIME2</i> , as an integer. If you select Rollover days/mins in the stream properties dialog box, a higher value of <i>TIME1</i> 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 <i>TIME1</i> causes the returned value to be negative.

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 upper case.

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.

Examples

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.

Note: @ functions cannot be called from scripts.

Available Sequence Functions

Function	Result	Description
MEAN(FIELD)	<i>Real</i>	Returns the mean average of values for the specified <i>FIELD</i> or <i>FIELDS</i> .
@MEAN(FIELD, EXPR)	<i>Real</i>	Returns the mean average of values for <i>FIELD</i> over the last <i>EXPR</i> records received by the current node, including the current record. <i>FIELD</i> must be the name of a numeric field. <i>EXPR</i> may be any expression evaluating to an integer greater than 0. If <i>EXPR</i> 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. Note that this function cannot be called from a script.
@MEAN(FIELD, EXPR, INT)	<i>Real</i>	Returns the mean average of values for <i>FIELD</i> over the last <i>EXPR</i> records received by the current node, including the current record. <i>FIELD</i> must be the name of a numeric field. <i>EXPR</i> may be any expression evaluating to an integer greater than 0. If <i>EXPR</i> 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. <i>INT</i> specifies the maximum number of values to look back. This is far more efficient than using just two arguments.
@DIFF1(FIELD)	<i>Real</i>	Returns the first differential of <i>FIELD1</i> . 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.
@DIFF1(FIELD1, FIELD2)	<i>Real</i>	The two-argument form gives the first differential of <i>FIELD1</i> with respect to <i>FIELD2</i> . Returns 0 if the relevant previous records do not exist.
@DIFF2(FIELD)	<i>Real</i>	Returns the second differential of <i>FIELD1</i> . 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.

Function	Result	Description
@DIFF2(FIELD1, FIELD2)	<i>Real</i>	The two-argument form gives the first differential of <i>FIELD1</i> with respect to <i>FIELD2</i> . Returns 0 if the relevant previous records do not exist.
@INDEX	<i>Integer</i>	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.
@LAST_NON_BLANK(FIELD)	<i>Any</i>	Returns the last value for <i>FIELD</i> that was not blank, according to any blank definition for <i>FIELD</i> 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 <i>FIELD</i> in the records read so far, \$null\$ is returned.
@MAX(FIELD)	<i>Number</i>	Returns the maximum value for the specified <i>FIELD</i> .
@MAX(FIELD, EXPR)	<i>Number</i>	Returns the maximum value for <i>FIELD</i> over the last <i>EXPR</i> records received so far, including the current record. <i>FIELD</i> must be the name of a numeric field. <i>EXPR</i> may be any expression evaluating to an integer greater than 0.
@MAX(FIELD, EXPR, INT)	<i>Number</i>	Returns the maximum value for <i>FIELD</i> over the last <i>EXPR</i> records received so far, including the current record. <i>FIELD</i> must be the name of a numeric field. <i>EXPR</i> may be any expression evaluating to an integer greater than 0. If <i>EXPR</i> is omitted, or if it exceeds the number of records received so far, the maximum value over all of the records received so far is returned. <i>INT</i> specifies the maximum number of values to look back. This is far more efficient than using just two arguments.
@MIN(FIELD)	<i>Number</i>	Returns the minimum value for the specified <i>FIELD</i> .
@MIN(FIELD, EXPR)	<i>Number</i>	Returns the minimum value for <i>FIELD</i> over the last <i>EXPR</i> records received so far, including the current record. <i>FIELD</i> must be the name of a numeric field. <i>EXPR</i> may be any expression evaluating to an integer greater than 0.
@MIN(FIELD, EXPR, INT)	<i>Number</i>	Returns the minimum value for <i>FIELD</i> over the last <i>EXPR</i> records received so far, including the current record. <i>FIELD</i> must be the name of a numeric field. <i>EXPR</i> may be any expression evaluating to an integer greater than 0. If <i>EXPR</i> is omitted, or if it exceeds the number of records received so far, the minimum value over all of the records received so far is returned. <i>INT</i> specifies the maximum number of values to look back. This is far more efficient than using just two arguments.

Function	Result	Description
@OFFSET(FIELD, EXPR)	<i>Any</i>	Retrieves values for a given field in the previous or following records. It returns the value of the field named <i>FIELD</i> in the record offset from the current record by the value of <i>EXPR</i> . If <i>EXPR</i> 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. For example, @OFFSET(Status, 1) returns the value of the Status field in the previous record, while @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. <i>EXPR</i> may also be an arbitrary CLEM expression, which is evaluated for the current record to give the offset. In this case, using the three-argument version of this function is recommended to improve performance (see below). If the expression returns anything other than a non-negative integer, this causes an error—that is, it is not legal to have calculated lookahead offsets.
@OFFSET(FIELD, EXPR, INT)	<i>Any</i>	Performs the same operation as the @OFFSET function with the addition of a third argument, <i>INT</i> , which specifies the maximum number of values to look back. In cases where the offset is computed from an expression, this third argument is recommended to improve performance. For example in an expression like @OFFSET(Foo, Month, 12) the system knows to keep only the last twelve values of Foo, otherwise it has to store every value just in case. For fixed offsets—including negative “lookahead” offsets, which must be fixed—the third argument is pointless, and the two-argument version of this function is recommended.
@SDEV(FIELD)	<i>Real</i>	Returns the standard deviation of values for the specified <i>FIELD</i> or <i>FIELDS</i> .
@SDEV(FIELD, EXPR)	<i>Real</i>	Returns the standard deviation of values for <i>FIELD</i> over the last <i>EXPR</i> records received by the current node, including the current record. <i>FIELD</i> must be the name of a numeric field. <i>EXPR</i> may be any expression evaluating to an integer greater than 0. If <i>EXPR</i> 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.

Function	Result	Description
@SDEV(FIELD, EXPR, INT)	<i>Real</i>	Returns the standard deviation of values for <i>FIELD</i> over the last <i>EXPR</i> records received by the current node, including the current record. <i>FIELD</i> must be the name of a numeric field. <i>EXPR</i> may be any expression evaluating to an integer greater than 0. If <i>EXPR</i> 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. <i>INT</i> specifies the maximum number of values to look back. This is far more efficient than using just two arguments.
@SINCE(EXPR)	<i>Any</i>	Returns the number of records that have passed since <i>EXPR</i> , an arbitrary CLEM expression, was true.
@SINCE(EXPR, INT)	<i>Any</i>	Adding the second argument, <i>INT</i> , specifies the maximum number of records to look back. If <i>EXPR</i> has never been true, <i>INT</i> is @INDEX+1.
@SINCE0(EXPR)	<i>Any</i>	Considers the current record, while @SINCE does not; @SINCE0 returns 0 if <i>EXPR</i> is true for the current record.
@SINCE0(EXPR, INT)	<i>Any</i>	Adding the second argument, <i>INT</i> specifies the maximum number of records to look back.
@SUM(FIELD)	<i>Number</i>	Returns the sum of values for the specified <i>FIELD</i> or <i>FIELDS</i> .
@SUM(FIELD, EXPR)	<i>Number</i>	Returns the sum of values for <i>FIELD</i> over the last <i>EXPR</i> records received by the current node, including the current record. <i>FIELD</i> must be the name of a numeric field. <i>EXPR</i> may be any expression evaluating to an integer greater than 0. If <i>EXPR</i> 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.
@SUM(FIELD, EXPR, INT)	<i>Number</i>	Returns the sum of values for <i>FIELD</i> over the last <i>EXPR</i> records received by the current node, including the current record. <i>FIELD</i> must be the name of a numeric field. <i>EXPR</i> may be any expression evaluating to an integer greater than 0. If <i>EXPR</i> 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. <i>INT</i> specifies the maximum number of values to look back. This is far more efficient than using just two arguments.
@THIS(FIELD)	<i>Any</i>	Returns the value of the field named <i>FIELD</i> in the current record. Used only in @SINCE expressions.

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

$(\text{Value} - @GLOBAL_MEAN(\text{Value})) / @GLOBAL_SDEV(\text{Value})$

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.

Note: @ functions cannot be called from scripts.

Function	Result	Description
@GLOBAL_MAX(FIELD)	<i>Number</i>	Returns the maximum value for <i>FIELD</i> over the whole data set, as previously generated by a Set Globals node. <i>FIELD</i> must be the name of a numeric field. If the corresponding global value has not been set, an error occurs. Note that this function cannot be called from a script.
@GLOBAL_MIN(FIELD)	<i>Number</i>	Returns the minimum value for <i>FIELD</i> over the whole data set, as previously generated by a Set Globals node. <i>FIELD</i> must be the name of a numeric field. If the corresponding global value has not been set, an error occurs.
@GLOBAL_SDEV(FIELD)	<i>Number</i>	Returns the standard deviation of values for <i>FIELD</i> over the whole data set, as previously generated by a Set Globals node. <i>FIELD</i> must be the name of a numeric field. If the corresponding global value has not been set, an error occurs.

Function	Result	Description
@GLOBAL_MEAN(FIELD)	<i>Number</i>	Returns the mean average of values for <i>FIELD</i> over the whole data set, as previously generated by a Set Globals node. <i>FIELD</i> must be the name of a numeric field. If the corresponding global value has not been set, an error occurs.
@GLOBAL_SUM(FIELD)	<i>Number</i>	Returns the sum of values for <i>FIELD</i> over the whole data set, as previously generated by a Set Globals node. <i>FIELD</i> must be the name of a numeric field. If the corresponding global value has not been set, an error occurs.

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.

Note: @ functions cannot be called from scripts.

Function	Result	Description
@BLANK(FIELD)	<i>Boolean</i>	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). Note that this function cannot be called from a script.
@LAST_NON_BLANK(FIELD)	<i>Any</i>	Returns the last value for <i>FIELD</i> that was not blank, according to any blank definition for <i>FIELD</i> 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 <i>FIELD</i> in the records read so far, \$null\$ is returned.
@NULL(FIELD)	<i>Boolean</i>	Returns true if the value of <i>FIELD</i> is the system-missing \$null\$. Returns false for all other values, including user-defined blanks. If you want to check for both, use @BLANK(FIELD) and @NULL(FIELD).
undef	<i>Any</i>	Used generally in CLEM to enter a \$null\$ value—for example, to fill blank values with nulls in the Filler node.

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, or to generate a list of fields as input. 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(@FIELD) derives a new log field for each selected field.

Note: @ functions cannot be called from scripts.

Function	Result	Description
@FIELD	<i>Any</i>	Performs an action on all fields specified in the expression context. Note that this function cannot be called from a script.
@TARGET	<i>Any</i>	When a CLEM expression is used in a user-defined analysis function, @TARGET represents the target field or “correct value” for the target/predicted pair being analyzed. This function is commonly used in an Analysis node.
@PREDICTED	<i>Any</i>	When a CLEM expression is used in a user-defined analysis function, @PREDICTED represents the predicted value for the target/predicted pair being analyzed. This function is commonly used in an Analysis node.
@PARTITION_FIELD	<i>Any</i>	Substitutes the name of the current partition field.
@TRAINING_PARTITION	<i>Any</i>	Returns the value of the current training partition. For example, to select training records using a Select node, use the CLEM expression: @PARTITION_FIELD = @TRAINING_PARTITION. This ensures that the Select node will always work regardless of which values are used to represent each partition in the data.
@TESTING_PARTITION	<i>Any</i>	Returns the value of the current testing partition.
@VALIDATION_PARTITION	<i>Any</i>	Returns the value of the current validation partition.

Function	Result	Description
@FIELDS_BETWEEN(start, end)	<i>Any</i>	Returns the list of field names between the specified start and end fields (inclusive) based on the natural (i.e., insert) order of the fields in the data. For more information, see “Summarizing Multiple Fields” in Chapter 7 on p. 121.
@FIELDS_MATCHING(pattern)	<i>Any</i>	Returns a list a field names matching a specified pattern. A question mark (?) can be included in the pattern to match exactly one character; an asterisk (*) matches zero or more characters. To match a literal question mark or asterisk (rather than using these as wildcards), a backslash (\) can be used as an escape character. For more information, see “Summarizing Multiple Fields” in Chapter 7 on p. 121.

Projects and Reports

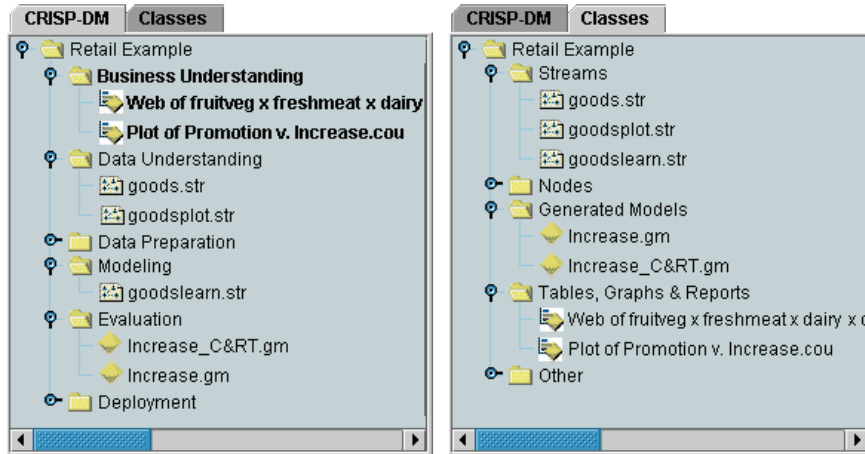
Introduction to Projects

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 choosing 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.

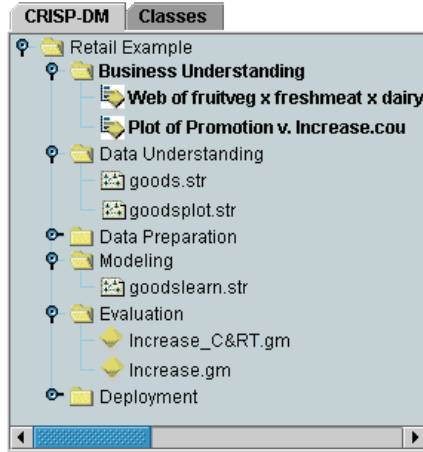
Figure 9-1
CRISP-DM view and Classes view of a project file



CRISP-DM View

By supporting the Cross-Industry Standard Process for Data Mining (CRISP-DM), Clementine projects provide an industry-proven and non-proprietary 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.

Figure 9-2
CRISP-DM view of the projects tool



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 the folder for the phase to set as the default.
- ▶ From the menu, choose Set as Default.

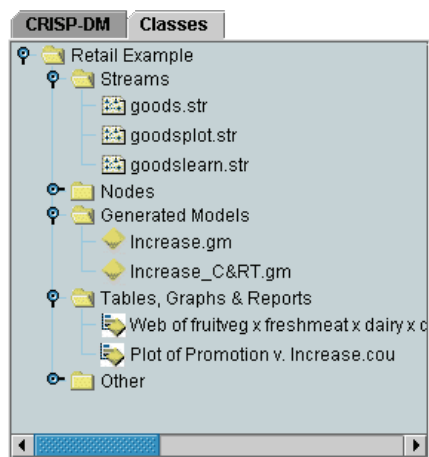
The default folder is displayed in bold type.

Classes View

The Classes view in the projects tool organizes your work in Clementine categorically by the types 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 9-3
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, nodes, and output objects) 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 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.

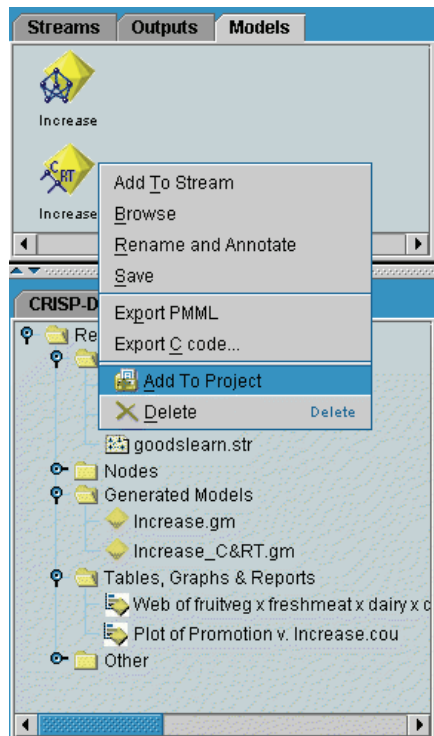
- Right-click and choose Add to Project.

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).

- Alternatively, you can drag and drop objects from the managers to the project workspace.

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 9-4
Adding items to a project



Adding Nodes from the Canvas

You can add individual nodes from the stream canvas by using the Save dialog box.

- ▶ Select a node on the canvas.
- ▶ Right-click and choose Save Node. Alternatively, from the menus 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 the target folder in CRISP-DM or Classes view.
- ▶ From the menu, choose Add to Folder.
- ▶ Select a file in the dialog box and click Open.

This will add a reference to the selected object inside Clementine projects.

Transferring Projects to Predictive Enterprise Repository

Note: A separate license is required to access this component. For more information, contact your sales representative or see the Clementine Web page (<http://www.spss.com/clementine/>).

You can transfer an entire project, including all component files, to Predictive Enterprise Repository (PER) in one step. Any objects that are already in the target location will not be moved. This feature also works in reverse: you can transfer entire projects from PER to your local file system.

To transfer a project:

- ▶ Make sure that the project you want to transfer is open in the Projects tool. Right-click the root project folder and choose Transfer Project.
- ▶ If prompted, log into PER.
- ▶ Specify the new location for the project and click OK.

Setting Project Properties

You can customize a project's contents and documentation by using the project properties dialog box. To access project properties:

- ▶ Right-click an object or folder in the projects tool and choose Project Properties.
- ▶ Click the Project tab to specify basic project information.

Figure 9-5
Setting project properties

The screenshot shows a dialog box titled "CustomerData" with a close button (X) in the top right corner. Below the title bar is a search field with a magnifying glass icon and a help button (?). The main area contains the following fields and controls:

- Created:** A text field containing "October 26, 2005 2:16:52 PM EDT".
- Summary:** A text field containing "CRM data from September".
- Contents:** A table with two columns: "Project objects" and "Counts".

Project objects	Counts
Streams	4
Nodes	0
Supernodes	0
Generated models	1
Tables, graphs and reports	1
Other	0
TOTAL	6
- Save unsaved objects as:** Two radio buttons. The first is selected and labeled "files on the local file system". The second is labeled "objects in the Predictive Enterprise Repository".
- Update object references when loading project:** A checked checkbox.
- Tabs:** Three tabs labeled "Project" (selected), "Report", and "Annotations".
- Buttons:** "OK", "Cancel", "Apply", and "Reset" at the bottom.

Created. Shows the project's creation date (not editable).

Summary. You can enter a summary for your data mining project that will be displayed in the project report.

Contents. Lists the type and number of components referenced by the project file (not editable).

Save unsaved object as. If you have licensed Predictive Enterprise Repository, use this setting to tell Clementine where the project components will be saved by default. For projects whose components are stored in the repository, choose *objects in the Predictive Enterprise Repository*.

Update object 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 file will remove that object from the project.

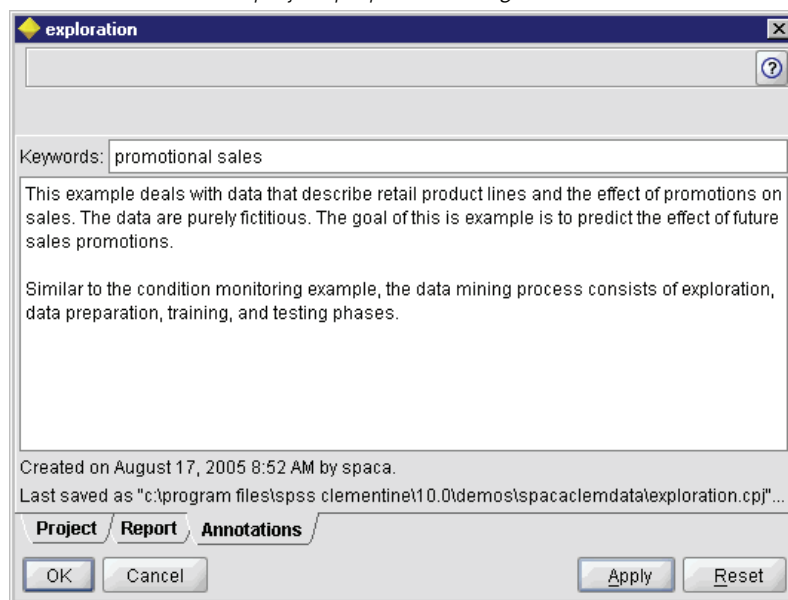
Annotating a Project

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:

- Click the Annotations tab.

Figure 9-6
Annotations tab in the project properties dialog box



- Enter keywords and text to describe the project.

Folder Properties and Annotations

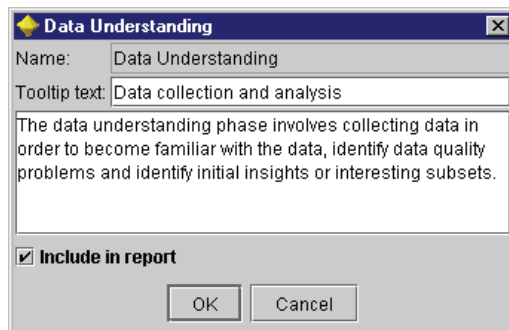
Individual project folders (in both CRISP-DM and Classes view) can be annotated. In CRISP-DM view, this can be an extremely effective way to document your organization's goals for each phase of data mining. For example, using the annotation tool for the *Business Understanding* folder, you can include 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 choose 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 9-7
Project folder with CRISP-DM annotation



Name. This area displays the name of the selected field.

Tooltip text. Create custom ToolTips that will be displayed when you hover the mouse pointer 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.

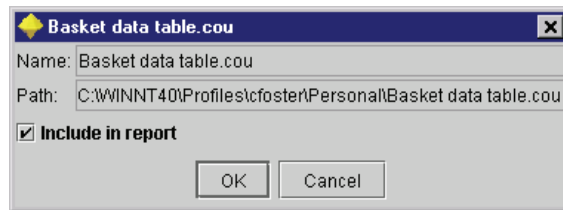
Include in report. To include the annotation in reports, select Include in report.

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 9-8
Object properties dialog box



Name. This area lists the name of the saved object.

Path. This area lists the location of the saved object.

Include in report. Select this option 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.
- ▶ If you are prompted to close or leave open all files associated with the project, click 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.

Generating a Report

One of the most useful features of projects is the ability to generate reports based on the project items and annotations. You can generate a report directly into one of several file types or to an output window on the screen for immediate viewing. From there, you can print, save, or view the report in a Web browser. You can distribute saved reports to others in your organization.

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 view or CRISP-DM view.

To generate a report:

- ▶ In the project properties dialog box, click the Report tab.
- ▶ Specify the report options and click Generate Report.

Figure 9-9
Selecting options for a report

The screenshot shows a dialog box titled "exploration" with a close button (X) in the top right corner. The dialog is divided into several sections:

- Report Format:**
 - Output name: ☐ Auto ☒ Custom (with a text field containing "Results Report")
 - ☒ Output to screen ☐ Output to file
 - Filename: ...
 - File type:
- Report Structure:**
 - Title:
 - Report structure:
 - Author:
 - Report includes:
 - ☐ only items marked for inclusion in the report
 - ☐ only items marked for exclusion from the report
 - ☒ all folders and objects
 - Select:
 - ☐ all items
 - ☒ recent items days old or less
 - ☐ old items days old or more
 - Order by:

At the bottom of the dialog, there is a "Generate Report" button with a green play icon. Below this are three tabs: "Project", "Report" (which is selected), and "Annotations". At the very bottom are four buttons: "OK", "Cancel", "Apply", and "Reset".

The options in the report dialog box provide several ways to generate the type of report you need:

Output name. Specify the name of the output window if you choose to send the output of the report to the screen. You can specify a custom name or let Clementine automatically name the window for you.

Output to screen. Select this option to generate and display the report in an output window. Note that you have the option to export the report to various file types from the output window.

Output to file. Select this option to generate and save the report as a file of the type specified in the File type list.

Filename. Specify a filename for the generated report. Files are saved by default to the Clementine *bin* directory. Use the ellipsis button (...) to specify a different location.

File type. Available file types are:

- **HTML document.** The report is saved as a single HTML file. If your report contains graphs, they are saved as PNG files and are referenced by the HTML file. When publishing your report on the Internet, make sure to upload both the HTML file and any images it references.
- **Text document.** The report is saved as a single text file. If your report contains graphs, only the filename and path references are included in the report.
- **Microsoft Word document.** The report is saved as a single document, with any graphs embedded directly into the document.
- **Microsoft Excel document.** The report is saved as a single spreadsheet, with any graphs embedded directly into the spreadsheet.
- **Microsoft PowerPoint document.** Each phase appears on a new slide. Any graphs are embedded directly into the PowerPoint slides.
- **Clementine output object.** When opened in Clementine, this file (.cou) is the same as the Output to screen option above.

Note: In order to export to a Microsoft Office file, you must have the corresponding application installed.

Title. Specify a title for the report.

Report structure. Select either CRISP-DM or Classes. CRISP-DM view provides a status report with “big-picture” synopses as well as details about each phase of data mining. Classes view 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 change it.

Report includes. 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 not selected).

Select. This option allows you to 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.

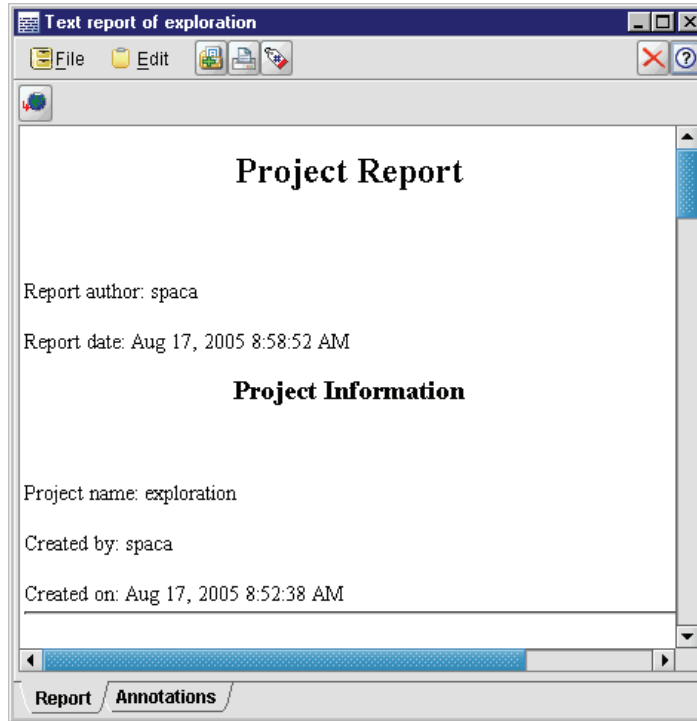
Order by. You can select a combination of the following object characteristics to order them within a folder:

- **Type.** Group objects by type.
- **Name.** Organize objects alphabetically.
- **Added date.** Sort objects using the date they were added to the project.

Saving and Exporting Generated Reports

A report generated to the screen is displayed in a new output window. Any graphs included in the report appear as in-line images. You can save, print, or export the report by using the toolbar buttons and menu options.

Figure 9-10
Generated report window



To save a report:

- ▶ From the File menu, choose Save.
 - ▶ Specify a filename.
- The report is saved as an output object.

To export a report:

- ▶ From the File menu, choose Export and the file type to which you want to export.
 - ▶ Specify a filename.
- The report is saved in the format you chose.

You can export to the following file types:

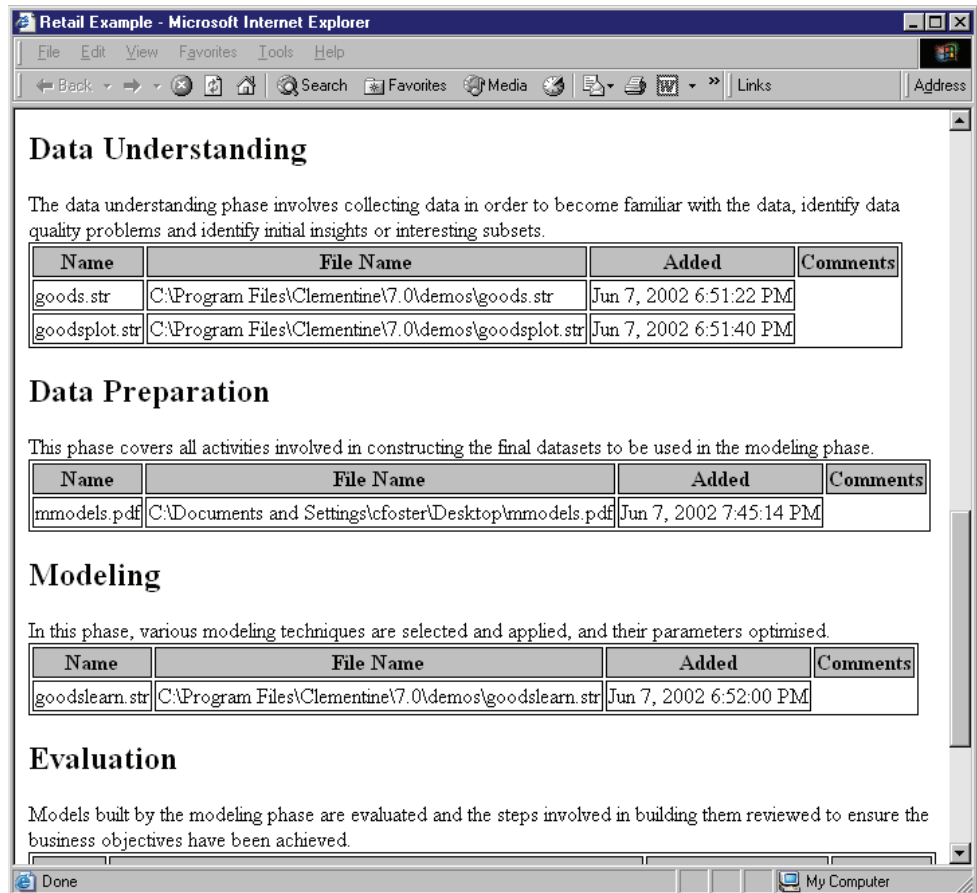
- HTML
- Text
- Microsoft Word
- Microsoft Excel
- Microsoft PowerPoint

Note: In order to export to a Microsoft Office file, you must have the corresponding application installed.

Use the buttons at the top of the window to:

- Print the report.
- View the report as HTML in an external Web browser.

Figure 9-11
Report displayed in a Web browser



Deploying 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 a number of ways to export the entire data mining process to external applications so that the work you do in Clementine to prepare data and build models can be used to your advantage outside of Clementine as well.

You can export, deploy, and manage models and streams in a number of ways:

- Use a Publisher node to export the stream and model for later use with Clementine Solution Publisher Runtime.
- Use the Predictive Applications Wizard to package the stream components for export to an external application such as PredictiveMarketing or another application. For more information, see “Predictive Applications Wizard” on p. 186.
- Use the Cleo Wizard to prepare a stream for deployment as a Cleo scenario for real-time scoring over the Web. For more information, see “Exporting to Cleo” on p. 196.
- Use SPSS Predictive Enterprise Repository to manage the life cycle of data mining models and related predictive objects in the context of enterprise applications, tools, and solutions. For more information, see “Predictive Enterprise Repository” in Chapter 11 on p. 203.

A separate license is required for each application. For more information, contact your sales representative or see the Clementine Web page (<http://www.spss.com/clementine/>).

Exporting Models as PMML

Alternatively, you can export one or more models in PMML, an XML-based format for encoding model information. For more information, see “Exporting Models as PMML” on p. 199.

Predictive Applications Wizard

The Predictive Applications Wizard allows you to package streams from Clementine for deployment with predictive applications, including PredictiveMarketing 2.0 and later. All of the data manipulation and modeling work done in Clementine can quickly be packaged and saved as a scoring solution. Once deployed, you can use the application to incorporate Clementine models into your campaign solution.

For example, a data analyst would like to segment her customers and develop a specialized value model for silver, gold, and platinum for use with PredictiveMarketing or another application. She can first develop the model in Clementine and then export it for use in PredictiveMarketing, which will use the Clementine model behind the scenes to recommend an offer that has the highest value.

Note: A separate license is required to access this component. For more information, contact your sales representative or see the Clementine Web page (<http://www.spss.com/clementine/>).

To access the wizard:

- From the menus choose:
 - Tools
 - Predictive Applications Wizard

Before Using the Predictive Applications Wizard

The following information is intended as an overview of the integration between Clementine and other predictive applications. Before you export a stream using the wizard, review the integration and prerequisites for publishing.

Note: A separate license is required to access this component. For more information, contact your sales representative or see the Clementine Web page (<http://www.spss.com/clementine/>).

How the Integration Works

Under normal circumstances, here's how you can use Clementine to expand the data mining and deployment capabilities available with predictive applications.

- ▶ Begin in your predictive application. Using options in the Customer View Builder, export the Unified Customer View (UCV) data model as an XML file. Note the location of the XML file because you will need it to guide your work in Clementine.
- ▶ Next, set up source nodes in Clementine to access all data sources (databases, flat files, etc.) that contain the fields for the UCV (these are listed in the XML file exported earlier). You may choose to include all fields referenced by the UCV, or you may use only the portion of the UCV that you will need for the models you are building. Typically, you will use several source nodes in Clementine to access the modeling data.
- ▶ Use Clementine to perform any data merging, transformations, or derivations that are necessary for your data mining task.
- ▶ At some point in the stream, be sure to include a Type node and name it *UCV*. This *UCV* Type node is used not only for directionality when modeling but also to ensure that your data matches the field information defined in the XML file. It is a good idea to compare settings in the Type node to attribute specifications in the XML file generated earlier. For more information, see “Step 3: Selecting a UCV Node” on p. 191.
- ▶ Next, consider the type of model you are creating in Clementine. If you are deploying a value model, such as a neural network, you may want to export binary predictions (for example, Churn True/False) as a propensity, which will make the prediction comparable with predictions from models generated by the application. For more information, see “Converting Scores to Propensities before Exporting” on p. 188.
- ▶ Once you are satisfied with the model and have converted confidences to propensities, add a Terminal node to the deployment branch of the stream. Many people use a Table node, but any Terminal node will suffice. Ensure that only fields you want visible in the outside application are visible at the Terminal node. In other words, prior to this Terminal node, you may need to filter out fields that you do not want to deploy.
- ▶ In addition, make sure that any prediction fields generated by the model are instantiated before being exported. If necessary, this can be done by adding a Type node between the generated model and the terminal “export” node.

- As a final step before using the wizard, ensure that your stream is prepared for deployment by performing a test execution.

The stream is now ready for deployment. You can access the Predictive Applications Wizard from the Tools menu in Clementine. Follow the wizard steps described in this documentation to produce a Clementine Deployment Package (.cdp) containing stream information and metadata required for publishing in the Real Time Environment.

Converting Scores to Propensities before Exporting

Before exporting with the Predictive Applications Wizard, consider the type of model that you have created in Clementine and the format of its scoring output.

- A **value model** creates a propensity score.
- An **offer model** creates prediction and confidence values that, when submitted to the application, will be used as “virtual attributes” in the UCV. These may be numeric or continuous values.

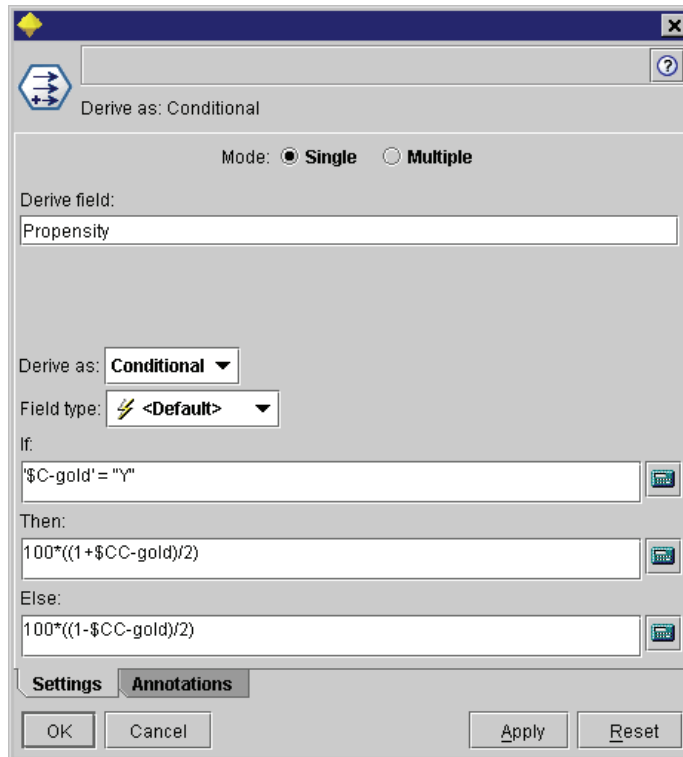
Note: A separate license is required to access this component. For more information, contact your sales representative or see the Clementine Web page (<http://www.spss.com/clementine/>).

If you are deploying a value model, such as a neural network, you may need to convert the prediction and confidence fields to a single probability field before exporting. This allows you to compare the strength of predictions accurately across several models. There are a number of methods that you may choose to convert confidences to propensities. A common approach for this conversion is as follows:

- In the scoring stream, attach a Derive node after the generated model.
- Set the Derive As option to Conditional and specify the following formula in the text boxes:

```
If '$C-gold' = "Y"  
Then 100*((1+$CC-gold)/2)  
Else 100*((1-$CC-gold)/2)
```

Figure 10-1
Derive node used to convert scores to propensity field



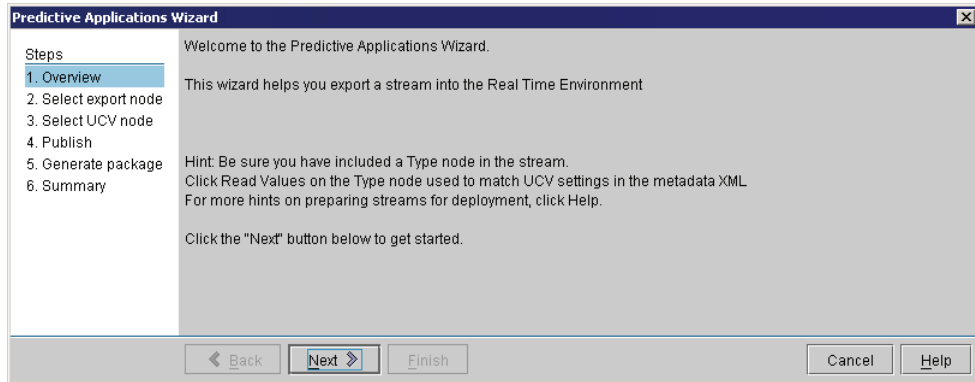
- Next, use a Filter node in the stream to remove the unnecessary generated fields *\$C-gold* and *\$CC-gold*.

Step 1: Predictive Applications Wizard Overview

When you first open the Predictive Applications Wizard, a welcome screen appears, orienting you to the process of bundling the necessary stream components.

Note: A separate license is required to access this component. For more information, contact your sales representative or see the Clementine Web page (<http://www.spss.com/clementine/>).

Figure 10-2
Welcome to the Predictive Applications Wizard

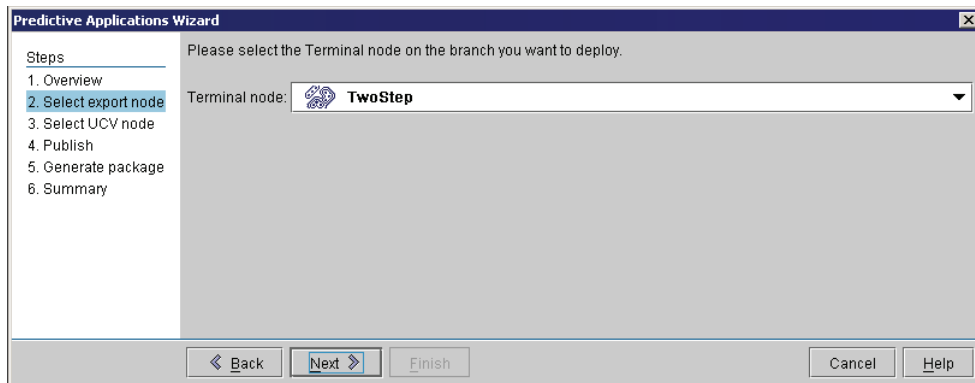


The rest of the wizard takes you through the process of generating a package for deployment into the Real Time Environment. Before proceeding, use the prerequisite checklist to ensure that the stream is prepared for deployment. For more information, see “Before Using the Predictive Applications Wizard” on p. 186.

Step 2: Selecting a Terminal Node

In step 2, you can specify a Terminal node from the stream that represents the scoring branch or the portion of the stream where scoring occurs.

Figure 10-3
Selecting a Terminal node



It is important to distinguish between Terminal nodes of the same name, since the drop-down list in the wizard provides only the name and node type for each Terminal node in the stream. To avoid confusion, give Terminal nodes unique names in the stream.

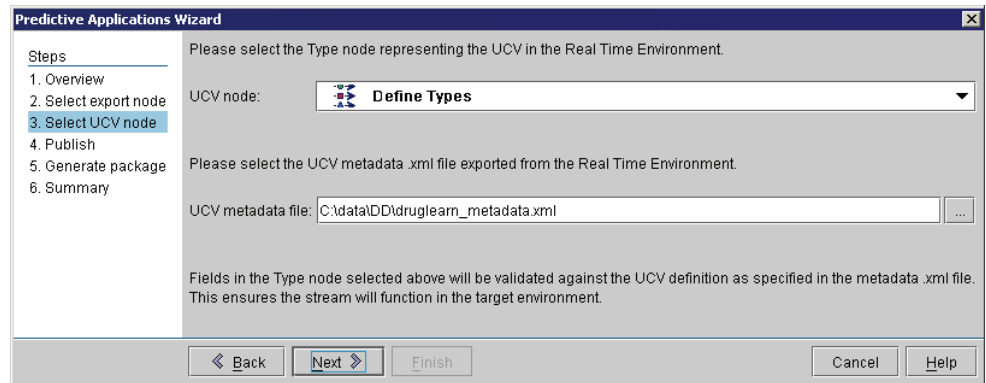
Also, ensure that only fields you want to be visible in the application environment are visible at the Terminal node. In other words, prior to this Terminal node, you may need to filter out fields that you do not want to deploy.

Step 3: Selecting a UCV Node

In Step 3, you specify two critical pieces of information for publishing—a UCV node and a UCV metadata file.

Figure 10-4

Selecting a Type node used as a UCV node and a UCV metadata file (XML file)



UCV node. A UCV node is a Type node from your stream that is used to ensure that all data matches the definitions in the Unified Customer View (UCV). These specifications are stored in an XML file (exported previously from the Customer View Builder). When you click Next, the wizard automatically validates settings in the Type node against specifications in the XML file that you specify here using the UCV metadata file option below.

UCV metadata file. A UCV metadata file is the XML file that you generated previously from the Customer View Builder. The XML file you choose here contains the data attributes required for deployment to the application.

Data Mismatch Errors

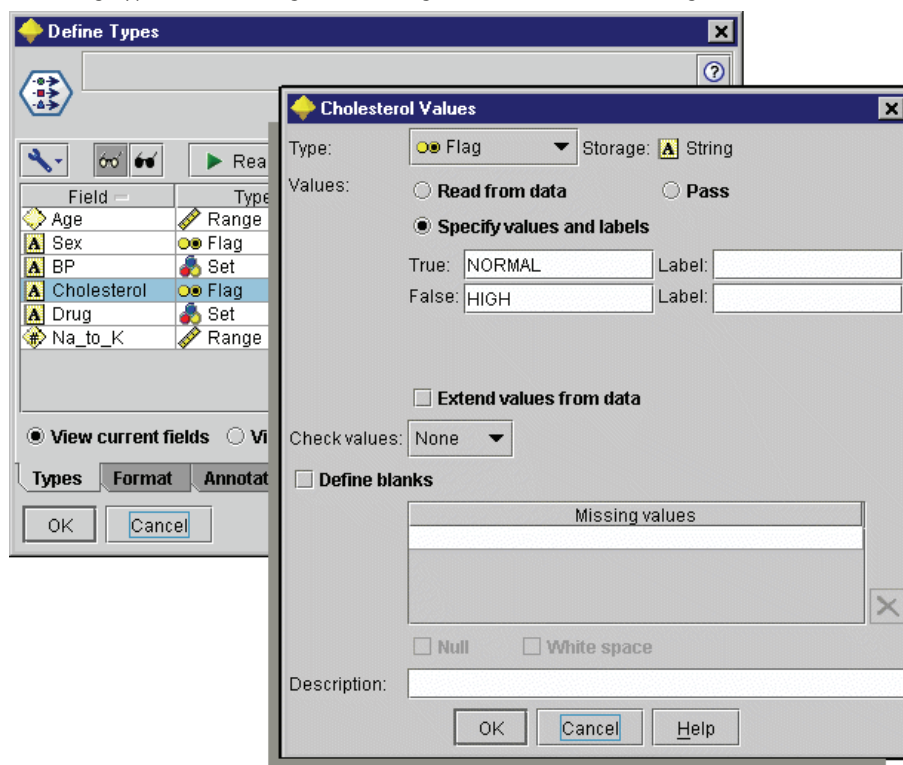
If the wizard has generated data mismatch errors, go back to the Clementine stream and examine your Type node specifications. Compare field information in the Type node to that in the XML file generated by Customer View Builder. (You can open the XML file in a text browser, such as Notepad). *Do your settings in the Type node match the UCV attributes?* For example, the XML file may state that a field named *Cholesterol* is required by the UCV and that it contains string values:

```
<UcvAttribute Name="Cholesterol" Domain="String"/>
```

In the Clementine stream, check the Type node settings to ensure that only fields useful to the Real Time Environment are exported from Clementine.

Figure 10-5

Checking Type node settings and storage in the Values subdialog box



Mapping Storage to Domain

In some cases, the mapping between Type node settings and UCV specifications is not obvious. In this integration, **domain** as defined in the UCV XML file is equivalent to **storage** in Clementine. Field settings for each of these will be matched during export.

Table 10-1
Storage and domain mapping

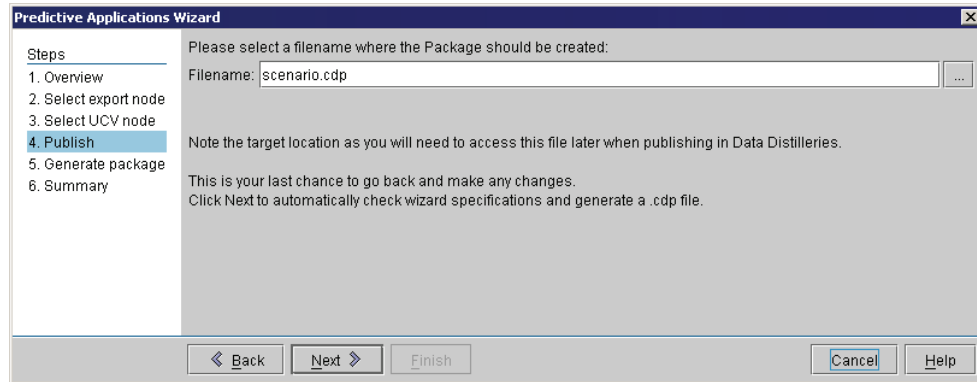
Clementine Storage	Application Domains
String	String Character Bit
Real	Float Double
Integer	Long Integer
Date	Date
Timestamp	Timestamp
Time	No match

You can alter storage type using conversion functions, such as `to_integer`, in a Derive or Filler node.

Step 4: Specifying a Package

In step 4, you must specify the name and location in which the stream package will be saved. Completing this step saves a number of objects as a bundle called a **Clementine Deployment Package**, which uses a `.cdp` extension.

Figure 10-6
Specifying a package name

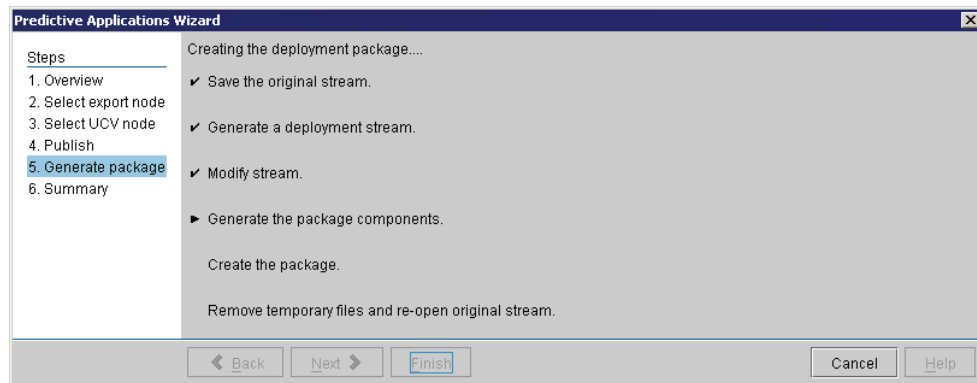


Click **Next** to automatically check the stream metadata and your specifications. If all are specified properly, the *.cdp* file will be generated. Note the target location because you will need to access this file later when publishing in the application.

Step 5: Generating the Package

At this stage, the wizard automatically checks the stream information and the specifications that you made in previous screens.

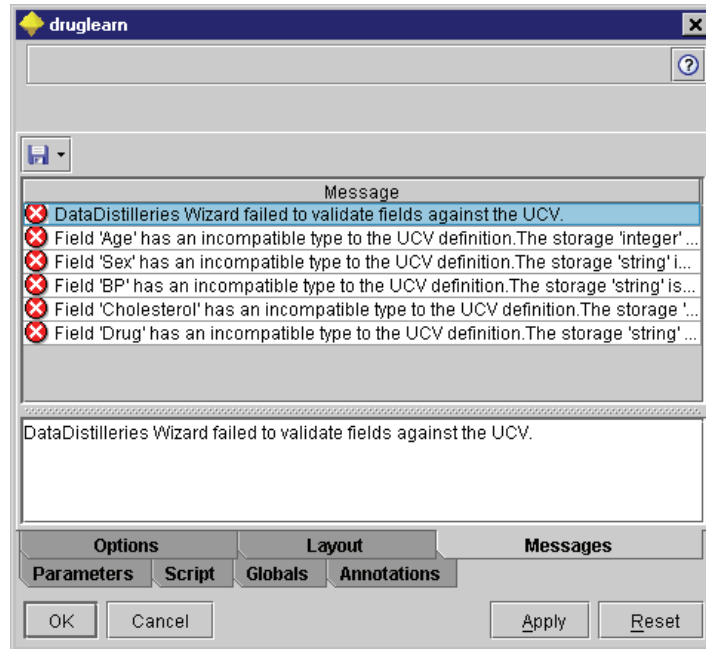
Figure 10-7
Creating the deployment package



Field names and types from the Clementine stream are verified against the XML file generated from the UCV. If field information does not match, the wizard may exit automatically and display relevant error messages.

Figure 10-8

Error messages resulting from metadata mismatch



In case of an error, return to the Clementine stream and check the following:

- Check that all field names in the UCV node (a Type node in the stream) are present in the XML file defining the UCV. Also check that their types (domains) are compatible.
- Note the case of field names, since both Clementine and other applications may be case sensitive. Field order, however, is not important.

Note: The size of the generated *.cdp* package needs to be under 5KB. In particular, if the list of generated fields is long, some of them might need to be removed from the description in order to stay within this limit.

Step 6: Summary

When your Clementine Deployment Package is successfully generated, you have finished your work in Clementine.

Taking the Next Step

Next, you can import the model (a *.cdp* package file) into Interaction Builder.

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 Wizard

This opens the Cleo Wizard, which takes you through the steps of specifying fields and other import information needed by Cleo.

Note: A separate license is required to access this component. For more information, contact your sales representative or see the Clementine Web page (<http://www.spss.com/clementine/>).

Cleo Wizard Overview

The Cleo Wizard enables you to perform two tasks:

- Create and deploy a Cleo scenario to the SPSS Web Deployment Framework (SWDF).
- Create and save a Cleo bundle (the ingredients for a Cleo scenario) to disk.

Note: A separate license is required to access this component. For more information, contact your sales representative or see the Clementine Web page (<http://www.spss.com/clementine/>).

What Is a Cleo Scenario?

A **Cleo scenario** is a Clementine data mining solution published to SWDF for use with Cleo. Selecting Publish Now in step 11 of the Cleo Wizard deploys the current stream and various specifications as a Cleo scenario ready for immediate use. Cleo end users can access the Cleo scenario to analyze data from a database or make predictions based on a single input record.

The HTML interface for a Cleo scenario is entirely customizable, 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 Cleo Bundle?

A **Cleo bundle** contains all of the components of a Cleo scenario without actually publishing to the SPSS Web Deployment Framework. Selecting Save Scenario in step 11 of the Cleo Wizard creates a *.jar* file containing the necessary ingredients for a Cleo scenario, which you can publish or alter at a later date. For example, the Cleo bundle includes stream and data specifications as well as a blueprint for the look of Cleo Web pages.

To modify a Cleo scenario, you can open the Cleo bundle using the Cleo Wizard and make any changes on the Wizard pages. To apply your new specifications, republish or save the bundle.

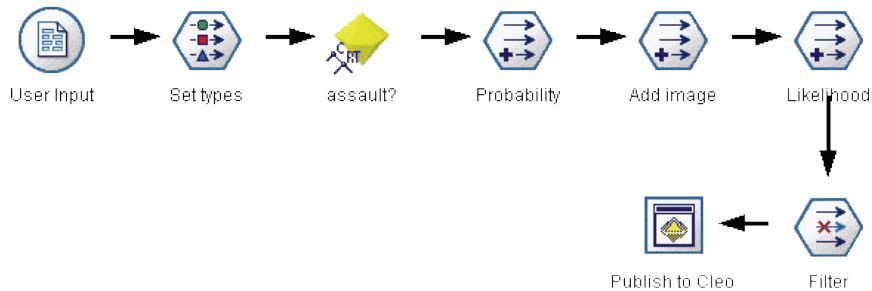
Cleo Stream Prerequisites

To optimize use of the Cleo Wizard and to ensure that your stream is prepared for deployment, consider the following recommendations before proceeding with the Wizard:

- Unlock the stream and save all changes before using the Cleo 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 formatting and other metadata used to describe the data and stream operations.

Figure 10-9

Example deployment stream with *Type* node and *Publisher* node



- Terminate the stream with a Publisher node. Make sure that all settings are complete in the Publisher node dialog box and deselect the Quote strings option.
- Perform a test execute to ensure that the stream is fully functional.
- Check that any ODBC connections used and mapped drives are available on the server machine.
- Always save a scenario—as well as publish it—to ensure that you can restore settings in the event of an error.

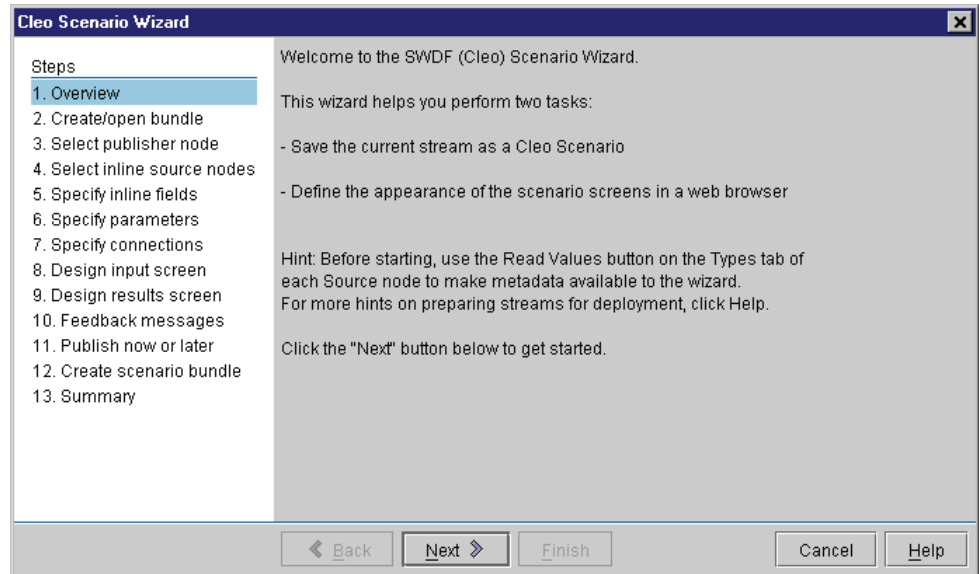
Note: A separate license is required to access this component. For more information, contact your sales representative or see the Clementine Web page (<http://www.spss.com/clementine/>).

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.

Note: A separate license is required to access this component. For more information, contact your sales representative or see the Clementine Web page (<http://www.spss.com/clementine/>).

Figure 10-10
Welcome to the Cleo Wizard



The Cleo Wizard walks you through the process of generating a bundle for deployment into the SPSS Web Deployment Framework. For additional hints on each screen, click the Help button to open the relevant topic in the online Help.

Before proceeding, you may want to use the prerequisite checklist to ensure that the stream is prepared for deployment. For more information, see “Cleo Stream Prerequisites” on p. 197.

Exporting Models as PMML

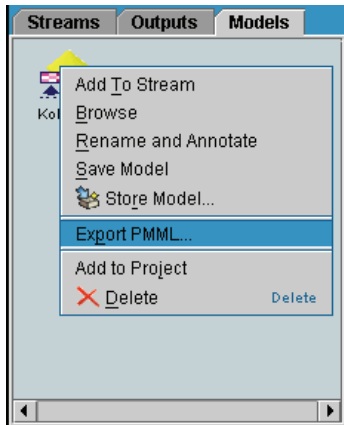
Models generated in Clementine can be exported as predictive model markup language, an XML format that makes it possible to share models with other applications that also support this format, such as SPSS. Clementine can also import models saved as PMML. For more details about PMML, see the data mining group Web site (<http://www.dmg.org>).

To export a model:

- Right-click a model on the Models tab in the managers window.

- From the context menu, choose Export PMML.

Figure 10-11
Exporting a generated model



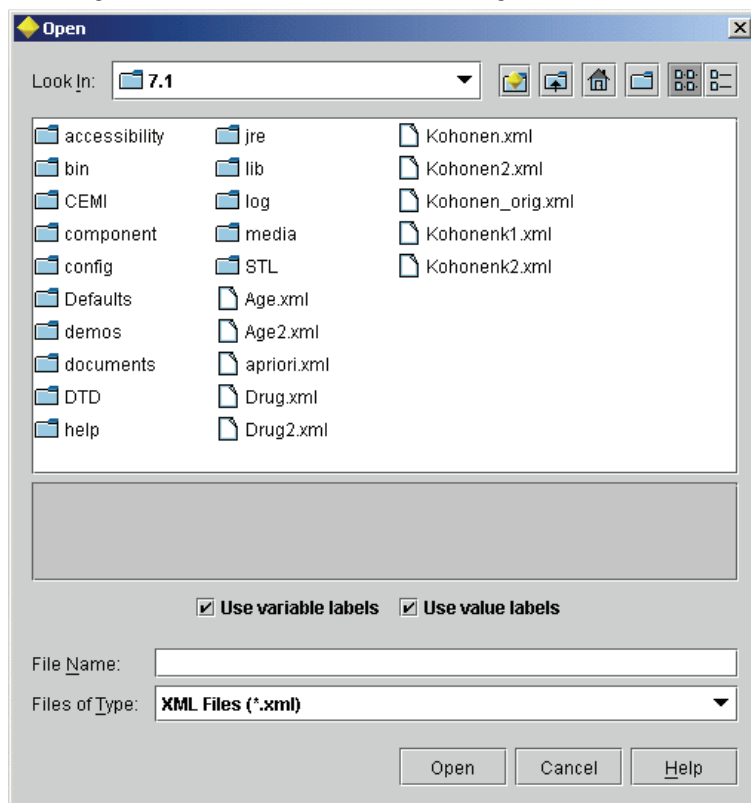
- In the Export dialog box, specify a target directory and a unique name for the model.

Note: You can change options for PMML export in the User Options dialog box. For more information, see “Setting PMML Export Options” in Chapter 3 on p. 30.

Importing Models Saved as PMML

Models exported as PMML from Clementine, SPSS, or another application can easily be brought into the generated models palette of Clementine. The context menu on the generated models palette provides an Import PMML option that opens a dialog box with options for doing this.

Figure 10-12
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, the variable names and literal values are used as normal. By default, both options are selected.

Model Types Supporting PMML

PMML import and export is supported for Clementine model types as follows. Note that all models are exported as PMML 3.0; when importing models, both PMML versions 2.1 and 3.0 are supported.

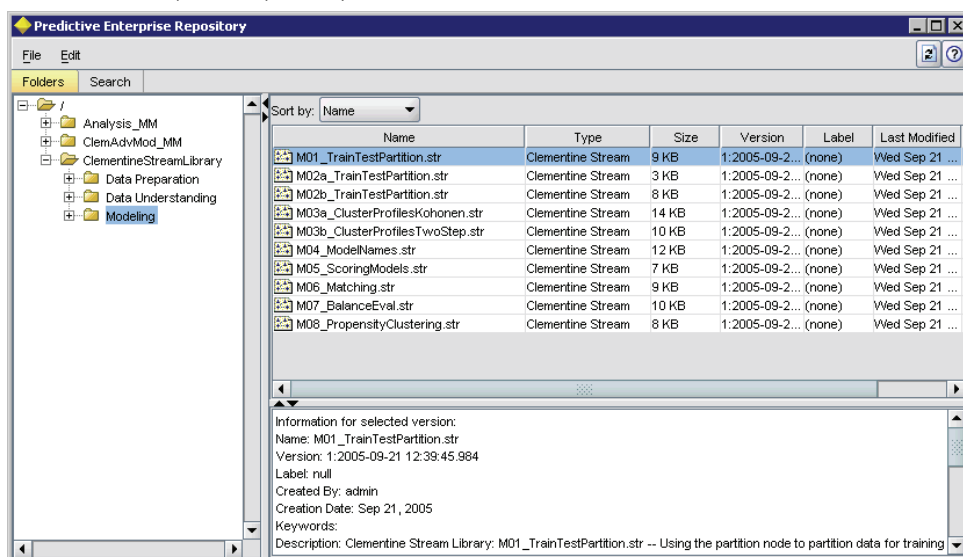
Model type	PMML Export (version 3.0)	PMML Import (2.1/3.0)
Neural Network	X	
C&R Tree	X	X
CHAID Tree	X	X
QUEST Tree	X	X
C5.0 Tree	X	
Ruleset	X	
Kohonen Net	X	
K-Means	X	
TwoStep	X	X
Linear Regression	X	X
Logistic Regression	X	X
Factor/PCA	X	
Sequence	X	
CARMA	X	
Apriori	X	
Text Extraction	not available	
Feature Selection	not available	
Anomaly Detection	not available	
Unrefined (GRI, CEMI)	not available	

Predictive Enterprise Repository

SPSS Predictive Enterprise Repository allows you to manage the life cycle of data mining models and related predictive objects in the context of enterprise applications, tools, and solutions. Clementine supports publishing and sharing of objects through Predictive Enterprise Repository, including streams, nodes, projects, models, and output files (*.cou). Objects are stored in a central repository, where they can be shared with other applications and tracked using extended versioning, metadata, and search capabilities.

Note: A separate license is required to access this component. For more information, contact your sales representative or see the Clementine Web page (<http://www.spss.com/clementine/>).

Figure 11-1
Predictive Enterprise Repository



Extended Versioning and Search Support

For example, suppose you create a stream and store it in the repository, where it can be shared with researchers from other divisions. If you later update the stream, you can add it to the repository without overwriting the previous version. All versions remain accessible and can be searched by name, label, fields used, or other attributes. For example, you could search for all model versions that use net revenue as a predictor or all models created by a particular author. (To do this with a traditional file system, you would have to save each version under a different filename, and the relationships between versions would be unknown to the software.)

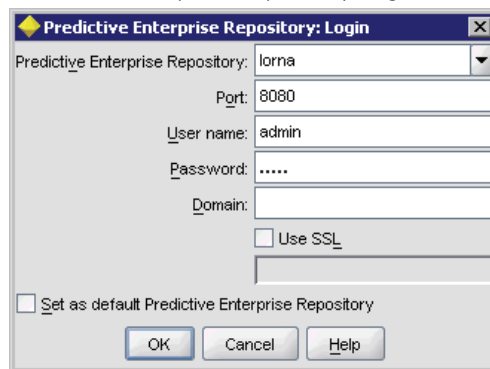
Connecting to Predictive Enterprise Repository

- ▶ To connect to Predictive Enterprise Repository, from the Clementine menus choose:
Tools
Predictive Enterprise Repository
Options...
- ▶ Specify login options as desired.

Settings are specific to each site or installation. For specific port, password, and domain information, contact your local system administrator.

Note: A separate license is required to access this component. For more information, contact your sales representative or see the Clementine Web page (<http://www.spss.com/clementine/>).

Figure 11-2
Predictive Enterprise Repository Login



Predictive Enterprise Repository. The Predictive Enterprise Repository installation you want to access. Generally, this matches the name of the host server where the repository is installed. You can connect to only one repository at a time.

Port. The port used to host the connection, typically 8080 by default.

User name and password. Specify a valid user name and password for logging in. In many cases, this may be the same password you use to log in to the local network. If necessary, contact your local administrator for more information.

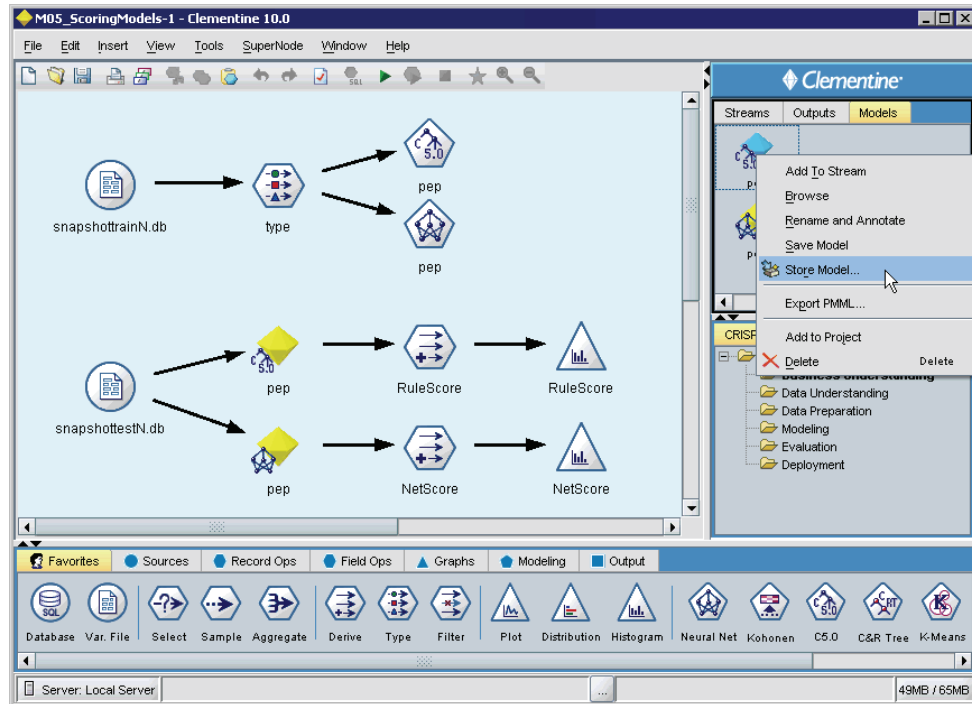
Domain. The network domain where the user is defined—for example, AD/SPSS or LDAP/SPSS, where the specified prefix matches the ID configured for the provider. Unless you are using a Windows Active Directory or LDAP domain, this field can typically be left blank. Contact your local administrator for specific login information if necessary.

Set as default Predictive Enterprise Repository. Saves the current settings as the default so that you do not have to reenter them each time you want to connect. *Note:* You must reenter the password whenever you open a new connection.

Use SSL. Specifies whether an SSL (**Secure Sockets Layer**) connection should be used. SSL is a commonly used protocol for securing data sent over a network. To use this feature, SSL must be enabled on the server hosting Predictive Enterprise Repository.

Storing Objects

Figure 11-3
Storing a model



You can store streams, nodes, models, projects, and output objects.

- ▶ To store the current stream, from the Clementine menus choose:
File
Store Stream...
- ▶ To store a model, project, or output object, select it on the manager palette in Clementine, and from the menus choose:
File
Models
Store Model...

or

File

Projects

Store Project...

or

File

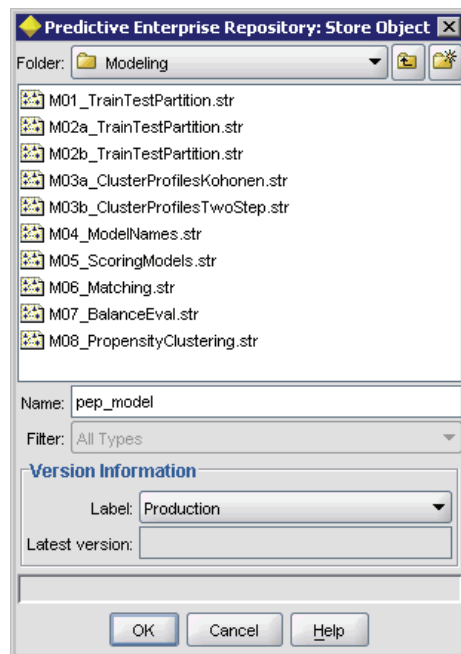
Outputs

Store Output...

- ▶ Alternatively, right-click on an object in the manager palette and choose Store.
- ▶ To store a node, right-click the node in the stream canvas and choose Store Node.

Figure 11-4

Storing a model



- ▶ Select the folder where you want to store the object. (To create a new folder, click the icon in the upper right corner.)
- ▶ Specify a name for the new object. To add a version, select the existing object. (Previous versions of the object will remain available.)

- Specify a label if desired. Choose <Add new label> to add a label, or select an existing label to reuse.

Labels can be moved from one version of an object to another. For example, you could label the current “production” version, and then reuse that label with a newer version when it becomes available. Note that only one version may carry a particular label at once.

Storing Projects

Because a project file is a container for other Clementine objects, you need to tell Clementine where to store the project’s objects—in the local file system or in Predictive Enterprise Repository. You do this using a setting in the Project Properties dialog box. Once you configure a project to store objects in the repository, whenever you add a new object to the project, Clementine automatically prompts you to store the object.

When you have finished your Clementine session, you must store a new version of the project file so that it remembers your additions. The project file automatically contains (and retrieves) the latest versions of its objects. So if you did not add any objects to a project during a Clementine session, then you do not have to restore the project file. You must, however, store new versions for the project objects (streams, output, and so forth) that you changed.

Retrieving Stored Objects

You can retrieve streams, models, nodes, projects, and output objects that have been stored in Predictive Enterprise Repository.

- To retrieve a stream, from the Clementine menus choose:
File
 Retrieve Stream...
 - To retrieve a model, project, or output object, from the Clementine menus choose:
File
 Models
 Retrieve Model...
- or*
- File
 Projects
 Retrieve Project...

or

File

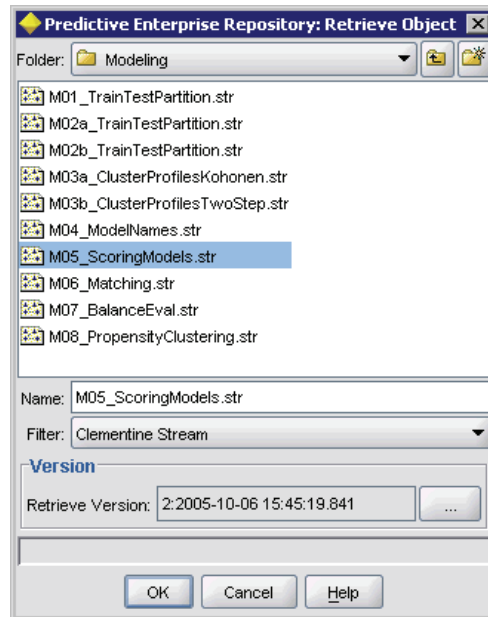
Outputs

Retrieve Output...

- ▶ Alternatively, right-click in the manager or project palette and choose Retrieve from the context menu.
- ▶ To retrieve a node, from the Clementine menus choose:
Insert
Node (or SuperNode) from Repository...

Figure 11-5

Retrieving an object from Predictive Enterprise Repository



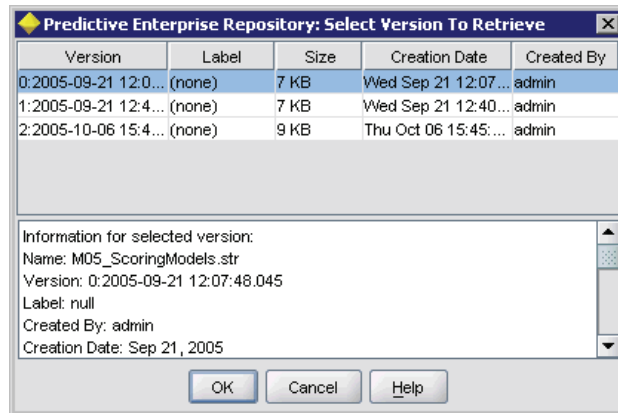
- ▶ Browse to select the object you want to retrieve.
- ▶ Select the desired object and version.

Selecting a Version

To retrieve a version other than the latest, click the version browse button (...). Detailed information for all versions is displayed, allowing you to choose the one you want.

Figure 11-6

Retrieving a version of an object



- To sort the list by version, label, size, or date, click on the header for the appropriate column.

You can access retrieved objects from the Stream, Model, or Output managers (as appropriate for the object type) in the Clementine window.

Browsing Repository Content

The Predictive Enterprise Repository allows you to browse stored content in a manner similar to Windows Explorer; you can also browse *versions* of each stored object.

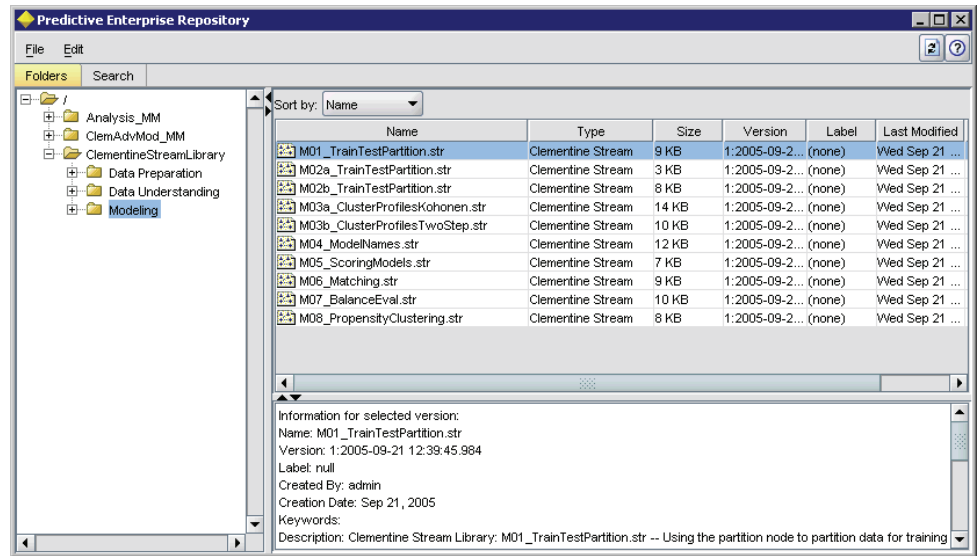
- To open the Predictive Enterprise Repository window, from the Clementine menus choose:

Tools

Predictive Enterprise Repository

Explore...

Figure 11-7
Exploring Predictive Enterprise Repository folders



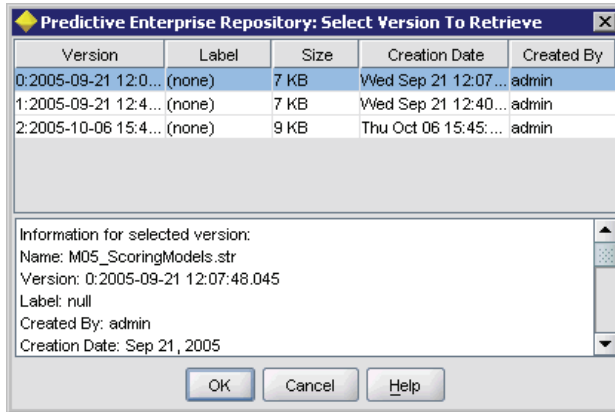
- To display a tree view of the folder hierarchy, click the Folders tab in the upper left pane.
- To locate stored objects by type, label, date, or other criteria, click the Search tab.

Objects that match the current selection or search criterion are listed in the right pane, with detailed information on the selected version displayed in the lower right pane. The attributes displayed apply to the most recent version.

- To browse or retrieve other versions of an object, click the object, and from the menus choose:

Edit
Retrieve Version...

Figure 11-8
Retrieving a version of an object



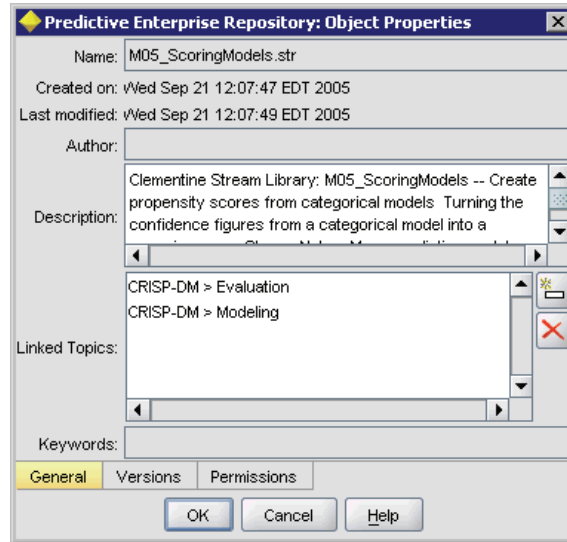
Object Properties

The Object Properties dialog box in Predictive Enterprise Repository allows you to view and edit properties. Although some properties cannot be changed, you can always update an object by adding a new version.

To view object properties:

- ▶ In the Predictive Enterprise Repository window, right-click the desired object.
- ▶ Choose Object Properties.

Figure 11-9
Object properties



General Tab

Name. The name of the object as viewed in Predictive Enterprise Repository.

Created on. Date the object (not the version) was created.

Last modified. Date the most recent version was modified.

Author. The user's login name.

Description. By default, this contains the description specified on the object's Annotation tab in Clementine.

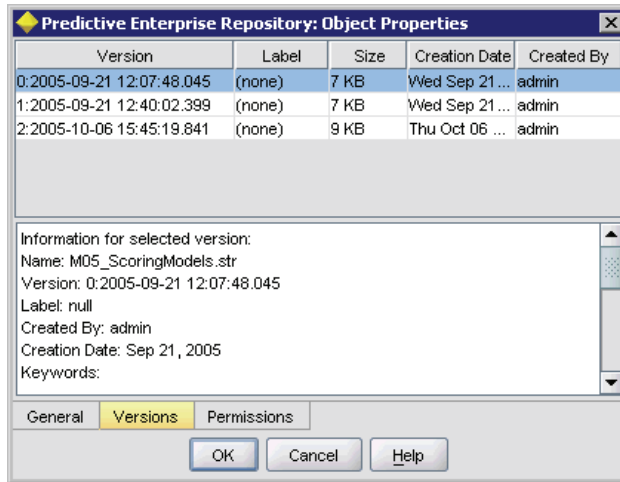
Linked topics. Predictive Enterprise Repository allows models and related objects to be organized by topics if desired. The list of available topics is provided by the local administrator.

Keywords. You specify keywords on the Annotation tab for a stream, model, or output object. Multiple keywords should be separated by spaces, up to a maximum of 255 characters. (If keywords contain spaces, use quotation marks to separate them.)

Versions Tab

Objects stored in Predictive Enterprise Repository may have multiple versions. The Versions tab displays information about each version.

Figure 11-10
Version properties



The following properties can be specified or modified for specific versions of a stored object:

Version. Unique identifier for the version, generated based on the time when the version was stored.

Label. Current label for the version, if any. Unlike the version identifier, labels can be moved from one version of an object to another.

The file size, creation date, and author are also displayed for each version.

Permissions Tab

The Permissions tab lets you set read and write permissions for the object. All users and groups with access to the current object are listed. Permissions follow a hierarchy. For example, if you do not have read permission, you cannot have write permission. If you do not have write permission, you cannot have delete permission.

Add group. Click the Add Group icon on the right side of the Permissions tab to assign access to additional users and groups. The list of available users and groups is controlled by the administrator.

Deleting Repository Objects

In order to delete an object from Predictive Enterprise Repository, you must decide if you want to delete all versions or just a particular version of an object.

To delete all versions of an object:

- ▶ In the Predictive Enterprise Repository window, right-click the desired object.
- ▶ Choose Delete Objects.

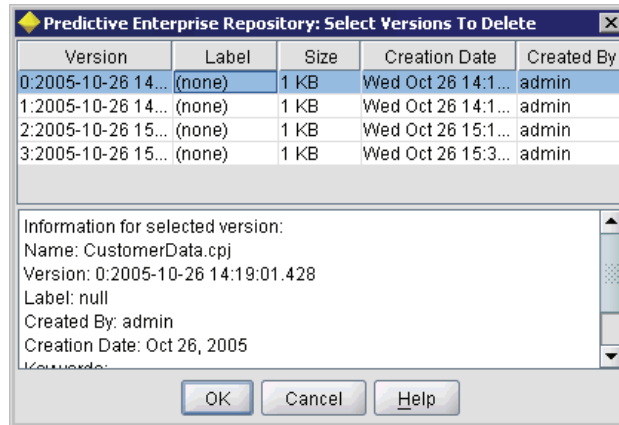
To delete the most recent version of an object:

- ▶ In the Predictive Enterprise Repository window, right-click the desired object.
- ▶ Choose Delete.

To delete a previous version of an object:

- ▶ In the Predictive Enterprise Repository window, right-click the desired object.
- ▶ Choose Delete Versions...
- ▶ Select the version(s) to delete and click OK.

Figure 11-11
Select Versions to Delete

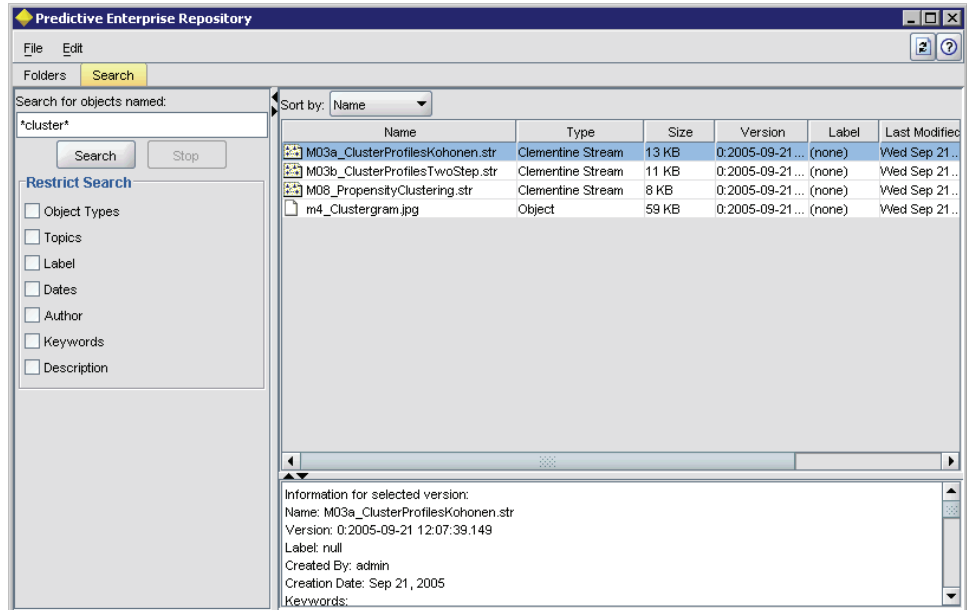


Searching Repository Content

You can search for objects by name, folder, type, label, date, or other criteria.

- ▶ From the Clementine menus choose:
 - Tools
 - Predictive Enterprise Repository
 - Explore...
- ▶ Click the Search tab.
- ▶ Specify the name of the object you want to find.

Figure 11-12
Searching objects by name



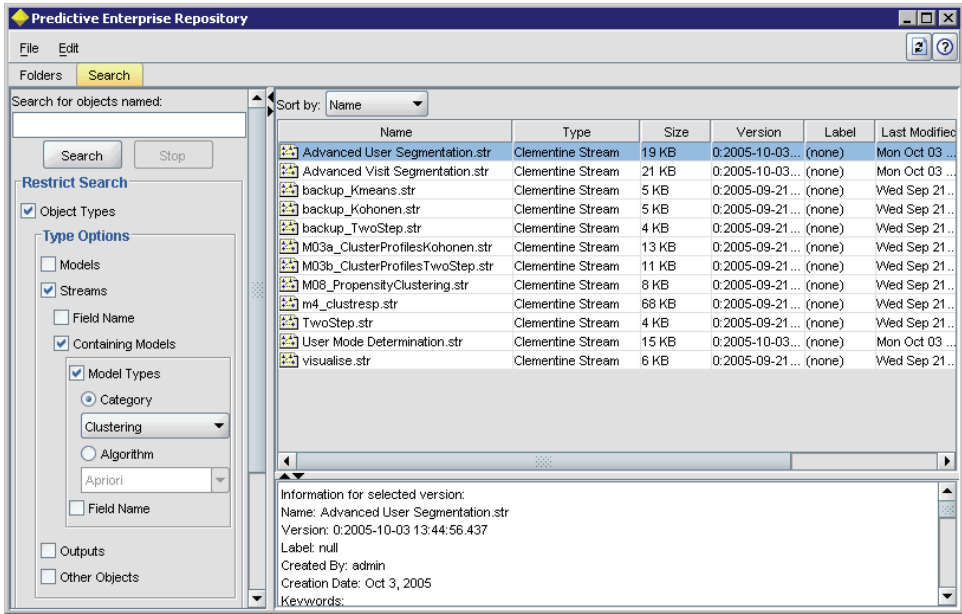
Searching objects by name. When searching on objects by name, an asterisk (*) can be used as a wildcard character to match any string of characters, and a question mark matches any single character. For example **cluster** matches all objects that include the string *cluster* anywhere in the name. The search string *m0?_** matches *M01_cluster.str* and *M02_cluster.str* but not *M01a_cluster.str*. Searches are not case-sensitive (*cluster* matches *Cluster* matches *CLUSTER*).

Note: If the number of objects is large, searches may take a few moments.

Refining the Search

You can refine the search based on object type, label, date, or keyword. Only objects that match *all* specified search criteria will be found. For example, you could locate all streams containing one or more clustering models that also have a specific label applied, and/or were modified after a specific date.

Figure 11-13
Searching for streams containing a specific type of model



Object Type. You can restrict the search to models, streams, output, or other types of objects (meaning *not* one of the other three).

- **Model Types.** You can search for models by category (classification, approximation, clustering, etc.) or by a specific model type, such as Kohonen.

Model Type	Model Name in Clementine
Classification	NN, Logistic, C5, C&RT, CHAID, QUEST Logistic
Approximation	Linear Regression, NN, C&RT (regression), CHAID (regression)
Clustering	Kohonen, Two Step, KMeans
Association	GRI, Apriori, Carma, Sequence
Sequence	Sequence
Reduction	Factor/PCA
Concept Extraction	Text Extraction

- **Field Name.** You can search by fields used—for example, all models that use a field named *income* as a target or predictor.

Streams. For streams, you can restrict the search by field name or model type.

Topics. You can search on models associated with specific topics. The list of available topics is provided by your local administrator.

Label. Restricts the search to specific labels.

Dates. You can specify a creation or modification date and search on objects before, after, or between the specified date range.

Keywords. Search on specific keywords. In Clementine, keywords are specified on the Annotation tab for a stream, model, or output object.

Description. Search on specific terms in the description field. In Clementine, the description is specified on the Annotation tab for a stream, model, or output object. Multiple search phrases can be separated by semicolons—for example, income; crop type; claim value. (Note that within a search phrase, spaces matter. For example, crop type with one space and crop type with two spaces are not the same.)

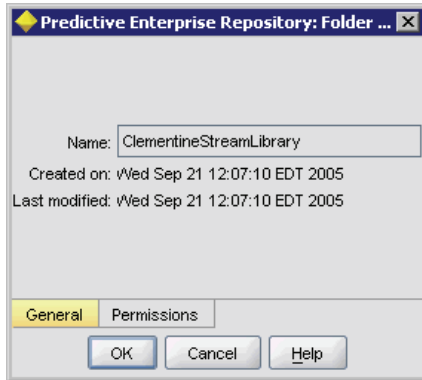
Adding and Removing Folders

- ▶ To add or remove folders in Predictive Enterprise Repository, from the Clementine menus choose:
 - Tools
 - Predictive Enterprise Repository
 - Explore...
- ▶ Click the Folders tab.
- ▶ To add a new folder, right-click the parent folder and choose New Folder.
- ▶ To delete a folder, right-click it and choose Delete Folder.
- ▶ To rename a folder, right-click it and choose Rename Folder.

Folder Properties

To view properties for any folder in the Predictive Enterprise Repository window, right-click the desired folder. Choose Folder Properties.

Figure 11-14
Folder properties



General tab. Displays the folder name, creation, and modification dates.

Permissions tab. Specifies read and write permissions for the folder. All users and groups with access to the parent folder are listed. Permissions follow a hierarchy. For example, if you do not have read permission, you cannot have write permission. If you do not have write permission, you cannot have delete permission.

- **Add group.** Click the Add Group icon on the right side of the Permissions tab to assign access to additional users and groups. The list of available users and groups is controlled by the administrator.
- **Cascade all permissions.** Cascades permissions settings from the current folder to all child and descendant folders. This is a quick way to set permissions for several folders at once. Set permissions as desired for the parent folder, and then cascade as desired.
- **Cascade changes only.** Cascades only changes made since the last time changes were applied. For example, if a new group has been added and you want to give it access to all folders under the Sales branch, you can give the group access to the root Sales folder and cascade the change to all subfolders. All other permissions to existing subfolders remain as before.
- **Do not cascade.** Any changes made apply to the current folder only and do not cascade to child folders.

Application Examples

Application Examples

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. You can load the data file(s) referenced for each application and follow the steps to learn the Clementine visual programming interface as well as data mining methods. The data files are available from the *Demos* directory of any Clementine Client installation and can be accessed from the Windows Start menu by choosing Start > [All] Programs > SPSS Clementine 10.0 > Demos.

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. Consulting the Clementine Application Templates (CATs), available on a separate CD from your SPSS representative, will provide a step-by-step guide to complex data mining applications.

Screening Predictors (Feature Selection)

The Feature Selection node helps you to identify the fields that are most important in predicting a certain outcome. From a set of hundreds or even thousands of predictors, the Feature Selection node screens, ranks, and selects the predictors that may be most important. Ultimately, you may end up with a quicker, more efficient model—one that uses fewer predictors, executes more quickly, and may be easier to understand. This example shows how a Feature Selection node can be used to build a model more efficiently.

The example data file is a data warehouse for a hypothetical telephone company. It contains information about responses to a special promotion by 5000 of the company's customers. The data includes a large number of fields containing customers' ages, employment, income, and telephone usage statistics. Three “target” fields show whether or not the customer responded to each of three offers. The company wants to use this data to help predict which customers are most likely to respond to similar offers in the future.

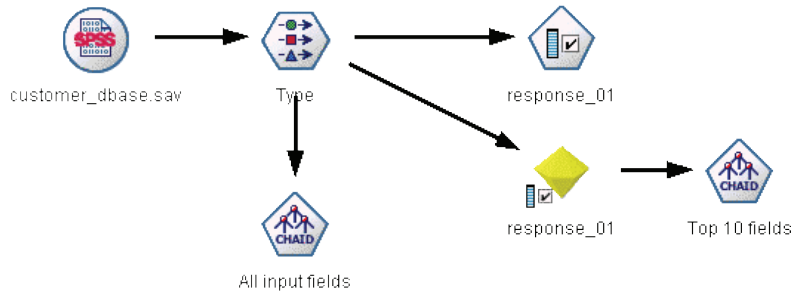
This example focuses on only one of the offers as a target. It uses the CHAID tree-building node to develop a model to describe which customers are most likely to respond to the promotion. It contrasts two approaches:

- Without feature selection. All predictor fields in the data set are used as inputs to the CHAID tree.
- With feature selection. The Feature Selection node is used to select the top 10 predictors. These are then input into the CHAID tree.

By comparing the two resulting tree models, we can see how feature selection produces effective results.

Building the Stream

Figure 13-1
Feature Selection example stream



- ▶ In a blank stream canvas, place an SPSS source node. Point this node to the example data file *customer_dbase.sav*, available in the *Demos* folder under your Clementine installation. (Alternatively, open the example stream file *featureselection.str* in the same directory.)
- ▶ Add a Type node and set the direction to *Out* for *response_01*. Set the direction to *None* for the customer ID (*custid*) and the other response fields (*response_02* and *response_03*). Leave the direction set to *In* for all other fields.
- ▶ Add a Feature Selection modeling node to the stream.
- ▶ Execute the stream to generate the feature selection model.

- Right-click the generated model in the Models manager and choose Browse to look at the results.

Figure 13-2
Feature Selection model

response_01

File Generate

Sort by: Rank

Rank	Field	Type	Importance	Value
20	spousedcat	Ordered ...	Important	1.000
21	cardmon	Range	Important	1.000
22	callcard	Flag	Important	0.999
23	addresscat	Ordered ...	Important	0.999
24	Intollmon	Range	Important	0.997
25	employ	Range	Important	0.995
26	card	Set	Important	0.995
27	voice	Flag	Important	0.984
28	lncardmon	Range	Important	0.982
29	pets_saltfish	Range	Important	0.974
30	cardten	Range	Important	0.972
31	wireless	Flag	Important	0.966
32	card2	Set	Important	0.965
33	commutecarpool	Flag	Important	0.954
34	commutepublic	Flag	Marginal	0.941
35	pets_reptiles	Range	Marginal	0.935
36	homeown	Flag	Marginal	0.906
37	agecat	Ordered ...	Marginal	0.904
38	ownpda	Flag	Marginal	0.903
39	birthmonth	Set	Unimport...	0.896

Selected fields:33 Total fields available:128

★ > 0.95 + <= 0.95 ▢ < 0.9

4 Screened Fields

Field	Type	Reason
ownstv	Flag	Single category too large
lnwireten	Range	Too many missing values
lnwiremon	Range	Too many missing values
lnequpm...	Range	Coefficient of variation below threshold

Model Summary Annotations

OK Cancel Apply Reset

The top panel shows the fields found to be useful in the prediction. These are ranked based on importance. The bottom panel shows which fields were screened from the analysis, and why. By examining the fields in the top panel, you can decide which ones to use in subsequent modeling sessions.

- ▶ To use the generated Feature Selection model, add it to the stream, connecting it to the Type node. Now double-click the node and use the model browser to select the fields to use downstream. Although 33 fields were originally identified as important, we want to reduce the set of predictors even further. So here, select only the top 10 predictors. (Use the check marks in the first column to deselect the unwanted predictors.)
- ▶ In order to compare results without feature selection, you must generate two models: one that uses feature selection and one that does not. Place two CHAID modeling nodes in the canvas, connecting one to the Type node and the other one to the generated Feature Selection model. In each CHAID node, turn on Interactive Tree mode in the node settings (Model tab).

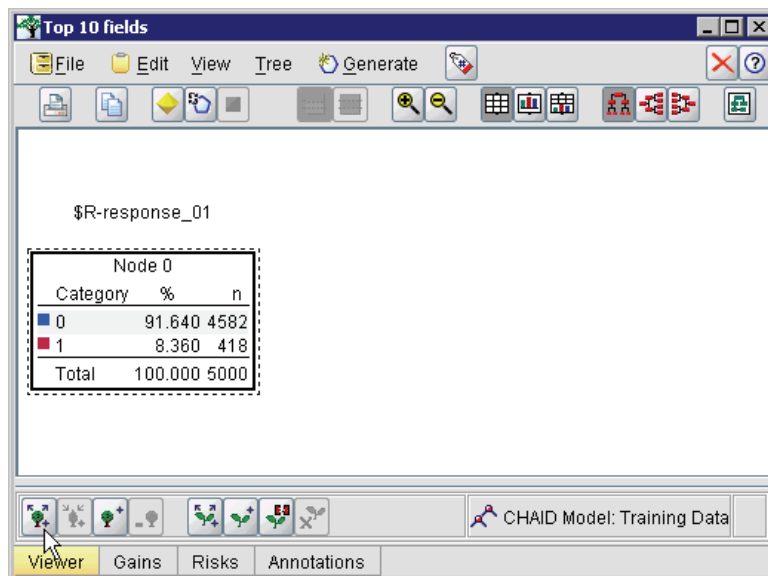
Building the Models

- ▶ Execute the CHAID node that uses all of the predictors in the data set (the one connected to the Type node). As it runs, notice how long it takes to execute. When the Tree Builder opens, click the Grow Tree button.

- Now do the same for the other CHAID node, which uses only 10 predictors. Again, grow the tree when the Tree Builder opens.

Figure 13-3

Growing the tree in the Tree Builder



The second model should have executed faster than the first one. Because this data set is fairly small, the difference in execution times is probably a few seconds; but for larger real-world data sets, the difference may be very noticeable—minutes or even hours. Using feature selection may speed up your processing times dramatically.

The second tree also contains fewer tree nodes than the first. It is easier to comprehend. But before you decide to use it, you need to find out whether it is effective and how it compares to the model that uses all predictors.

Comparing the Results

To compare the two results, we need a measure of effectiveness. For this, we can look at the Gains tab in the Tree Builder. We will look at **lift**, which measures how much more likely the records in a node are to fall under the target category when compared to all records in the data set. For example, a lift value of 148% indicates that records in

the node are 1.48 times more likely to fall under the target category than all records in the data set. Lift is indicated in the Index column on the Gains tab.

- In the Tree Builder for the full set of predictors, click the Gains tab. Change the Target Category to *1*. Change the display to quartiles by first clicking the Quantiles button. Then select Quartile from the menu next to this button.
- Repeat this procedure in the Tree Builder for the set of 10 predictors so that you have two similar Gains tables to compare, as shown in the following figures.

Figure 13-4

Gains table for the CHAID model with all predictors included

Nodes	Percentile	Percentile: n	Gain: n	Gain (%)	Response (%)	Index (%)
44,29,26,43,8,42,38,47,33,24,50	25.00	1250.00	230.00	55.04	18.41	220.16
50,21,22,56,53,41,40,46,48	50.00	2500.00	352.00	84.11	14.06	168.23
48,32,49,25,52,19,45	75.00	3750.00	401.00	95.94	10.69	127.92
45,23,54,37,39,35,51,55	100.00	5000.00	418.00	100.00	8.36	100.00

Figure 13-5

Gains table for the CHAID model with only the top 10 predictors included

Nodes	Percentile	Percentile: n	Gain: n	Gain (%)	Response (%)	Index (%)
21,9,15,12	25.00	1250.00	200.00	47.93	16.03	191.74
12,24,10,7	50.00	2500.00	307.00	73.36	12.27	146.71
7,11,18	75.00	3750.00	385.00	92.14	10.27	122.86
18,22,16,17,23	100.00	5000.00	418.00	100.00	8.36	100.00

Each Gains table groups the terminal nodes for its tree into quartiles. To compare the effectiveness of the two models, look at the lift (Index value) for the top quartile in each table.

When all predictors are included, the model shows a lift of 220%. That is, cases with the characteristics in these nodes are 2.2 times more likely to respond to the target promotion. To see what those characteristics are, click to select the top row. Then switch to the Viewer tab, where the corresponding nodes are now highlighted in black. Follow the tree down to each highlighted terminal node to see how the predictors were split. The top quartile alone includes 11 nodes. When translated into real-world scoring models, 11 customer profiles can be difficult to manage!

With only the top 10 predictors (as identified by feature selection) included, the lift is 192%. Although this model is not quite as good as the model that uses all predictors, it is certainly useful. And here the top quartile includes only four nodes, so it is simpler. Therefore, we can determine that the feature selection model is preferable to the one with all predictors.

Summary

Let's review the advantages of feature selection. Using fewer predictors is less expensive. It means that you have less data to collect, process, and feed into your models. And computing time is improved. In this example, even with the extra feature selection step, model-building was noticeably faster with the smaller set of predictors. With a larger real-world data set, the time savings should be greatly amplified.

Using fewer predictors results in simpler scoring. As the example shows, you might identify only four profiles of customers who are likely to respond to the promotion, rather than 11. And indeed, with larger numbers of predictors, you run the risk of overfitting your model. The simpler model may generalize better to other data sets (although you would need to test this to be sure).

You could have used a tree-building algorithm to do the feature selection work, allowing the tree to identify the most important predictors for you. In fact, the CHAID algorithm is often used for this purpose, and it is even possible to grow the tree level by level in order to control its depth and complexity. However, the Feature Selection node is faster and easier to use. It ranks all of the predictors in one fast step, allowing you to quickly identify the most important fields. It also allows you to vary the number of predictors to include. You could easily run this example again using the top 15 or 20 predictors instead of 10, comparing the results to determine the optimal model.

Fraud Screening (Anomaly Detection/NeuralNet)

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 anomaly detection to discover deviations from the norm, highlighting those records that are abnormal and worthy of further investigation. You are particularly interested in grant applications that appear to claim too much (or too little) money for the type and size of farm.

The analysis is conducted in two stages—a preliminary screening using Anomaly Detection, followed by a more in-depth exploration and modeling using a combination of methods.

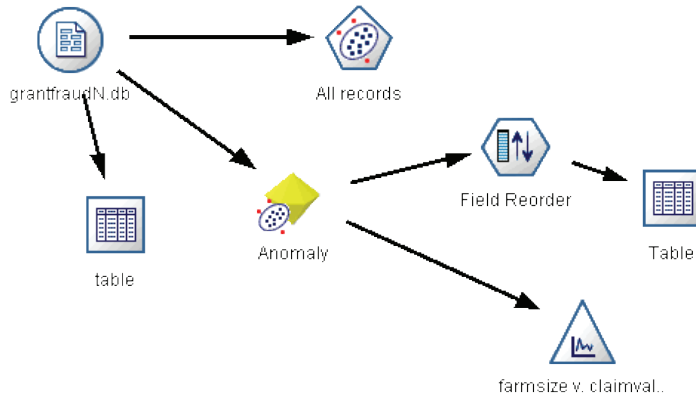
The data contains nine fields:

- *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.

Preliminary Screening

To do a quick screening for unusual records, you can use the Anomaly Detection node.

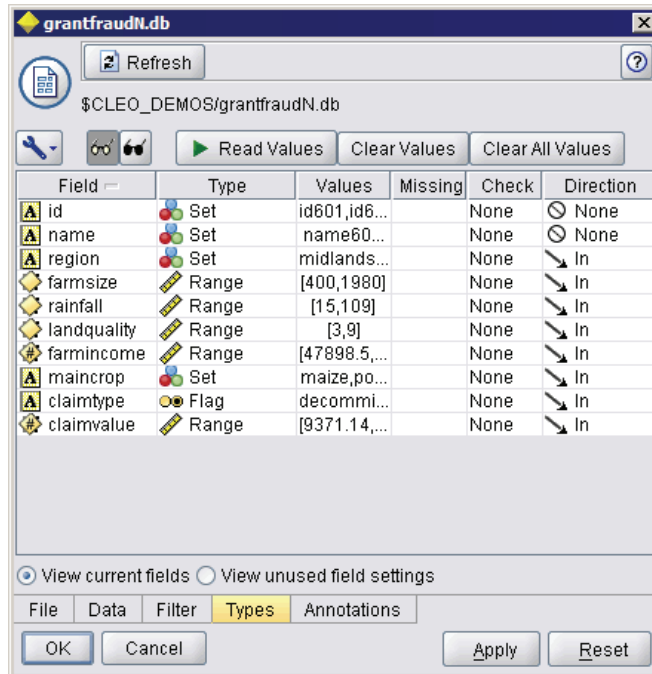
Figure 14-1
Preliminary screening for anomalies



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.

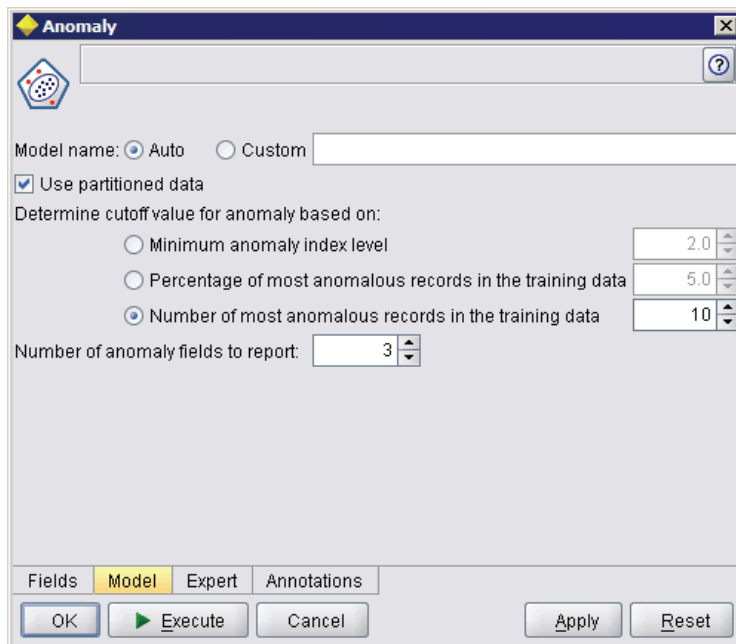
In the Type node, set the direction for the *name* and *ID* fields to *None*, since these fields are not useful in modeling. The direction for all other fields should be set to *In* so that they will be included as inputs to the Anomaly Detection model. (All other fields will be ignored.)

Figure 14-2
Setting field type and direction in the source node



On the Model tab of the Anomaly Detection node, select Number of most anomalous records in the training data, and enter 10 as the value. Then execute the node, and add the generated model to the stream.

Figure 14-3
Anomaly model settings



The screenshot shows the 'Anomaly' dialog box with the 'Model' tab selected. The 'Model name' is set to 'Auto'. The 'Use partitioned data' checkbox is checked. Under 'Determine cutoff value for anomaly based on:', the 'Number of most anomalous records in the training data' option is selected, with a value of 10. The 'Number of anomaly fields to report' is set to 3. The 'Fields' tab is also visible, showing a list of fields. The 'Execute' button is highlighted.

Anomaly

Model name: ☒ Auto ☐ Custom

☒ Use partitioned data

Determine cutoff value for anomaly based on:

- ☐ Minimum anomaly index level
- ☐ Percentage of most anomalous records in the training data
- ☒ Number of most anomalous records in the training data

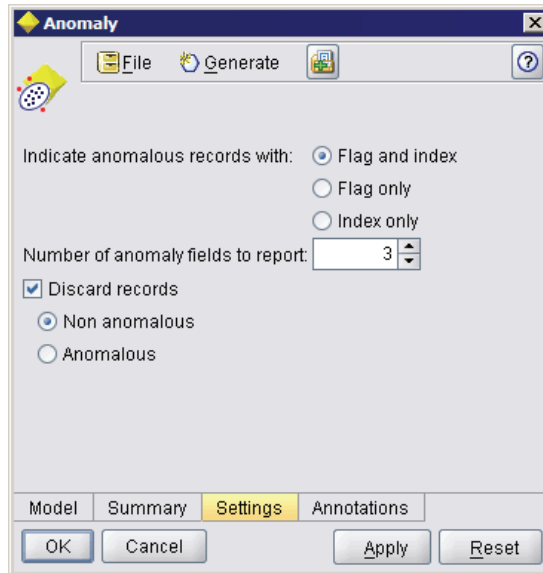
Number of anomaly fields to report:

Fields **Model** Expert Annotations

OK **Execute** Cancel Apply Reset

On the Settings tab of the generated model, you can select the option to discard non-anomalous records so that only records flagged as potential anomalies during scoring will remain in the stream.

Figure 14-4
Setting for applying the generated model



Attach a table node and execute to see the scored data. The *\$0-Anomaly* field generated by the model indicates which records are potential anomalies. Because the Discard records option was selected on the Settings tab, only records flagged as potential anomalies are listed. The overall anomaly index value is also listed for each record,

along with the peer group and the three fields most responsible for causing that record to be anomalous.

Figure 14-5

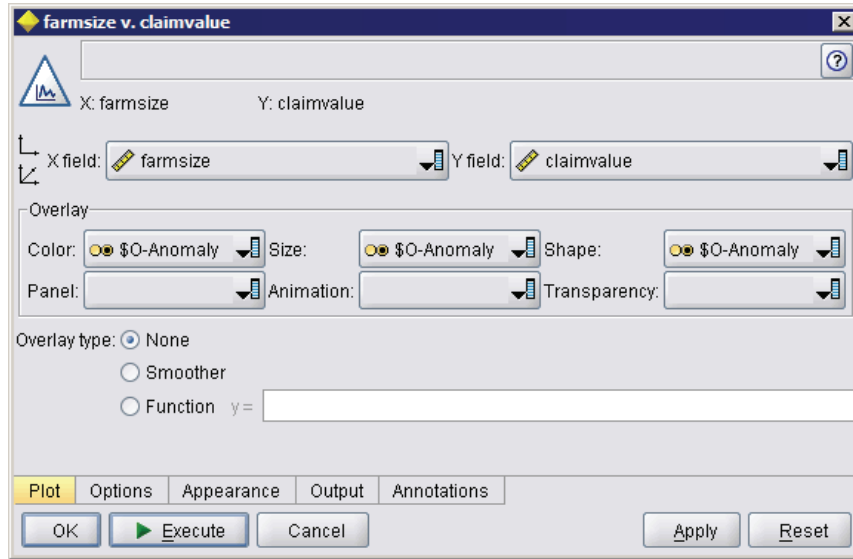
Table listing potential anomalies

	id	\$O-Anomaly	\$O-AnomalyIndex	\$O-PeerGroup	\$O-Field-1	\$O-FieldImpact-1	\$O-Field-2
1	id633	T	1.602	2	claimvalue	0.358	farminco...
2	id647	T	1.404	2	farminco...	0.334	claimvalue
3	id654	T	1.497	2	rainfall	0.322	maincrop
4	id703	T	1.359	1	rainfall	0.230	region
5	id704	T	1.428	2	farminco...	0.287	maincrop
6	id739	T	1.685	2	claimvalue	0.404	farminco...
7	id752	T	1.771	2	claimvalue	0.391	farminco...
8	id791	T	1.387	1	maincrop	0.236	rainfall
9	id813	T	1.643	1	region	0.181	landquality
10	id883	T	1.351	2	region	0.187	maincrop

You can use charts to gain a better picture of which records are being flagged. For example, you can plot *farmsize* against *claimvalue* and overlay the *\$O-Anomaly* field to see where the anomalous records fall. (Attach the Plot node to the generated Anomaly Detection model. To see the complete plot showing all records—not just

anomalies—deselect the Discard records option on the Settings tab in the generated model.)

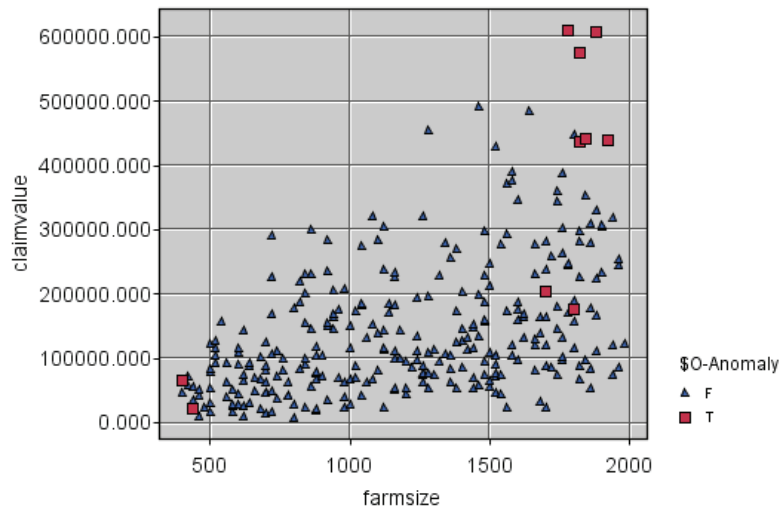
Figure 14-6
Creating the overlay plot



As you might expect, the most expensive claims are flagged. However, a number of other claims are also flagged, including some of the least expensive. To understand what is happening, it is worth taking a closer look at the data.

Figure 14-7

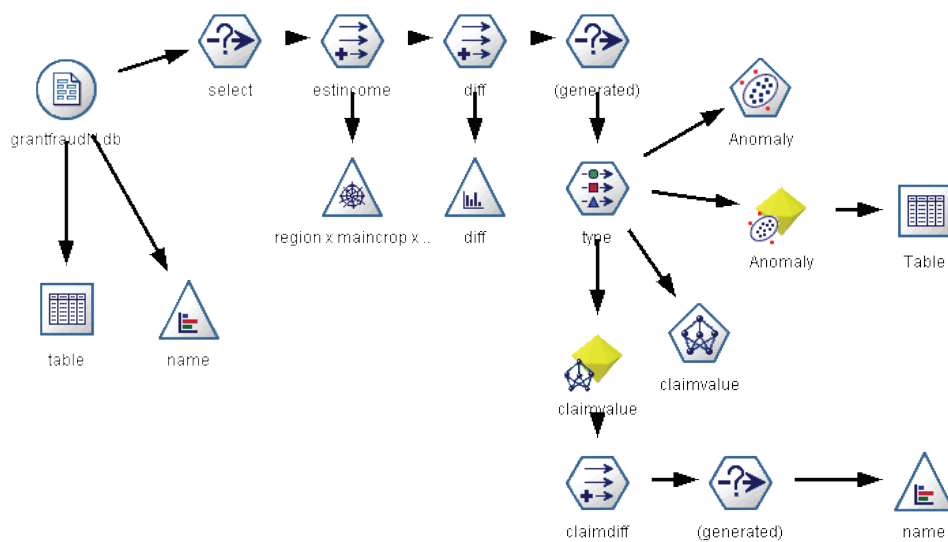
Anomalies plotted against farmsize and claimvalue



Data Investigation

To continue your analysis of the fraud data, you can investigate the data using exploratory graphics. This helps you to form hypotheses that can be useful in modeling. For this example, we'll work step by step, from accessing data through training a neural net.

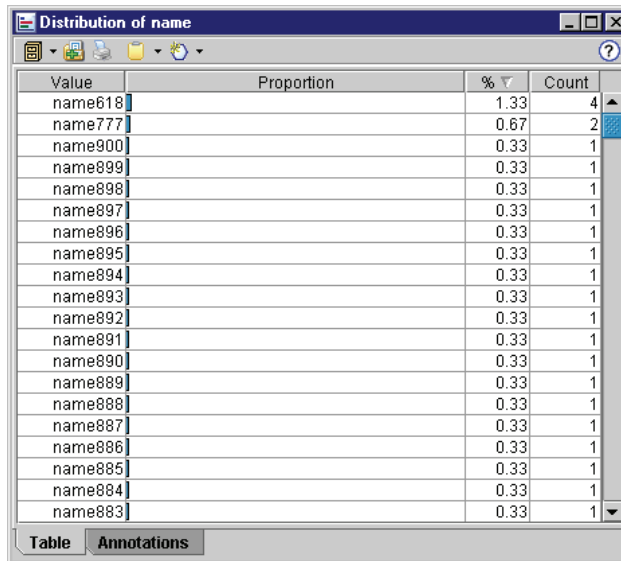
Figure 14-8
Stream diagram illustrating the operations of *fraud.str* stream



Initially, consider the possible types of fraud in the data. One such possibility is multiple grant aid applications from a single farm.

To check for duplicate claims, connect a Distribution node to the data set and select the *name* field, which is supposed to have a unique value identifying each farm. The resulting distribution plot shows that a few farms have made multiple claims. (To see this, click on the *Count* column header to sort rows by count in descending order.)

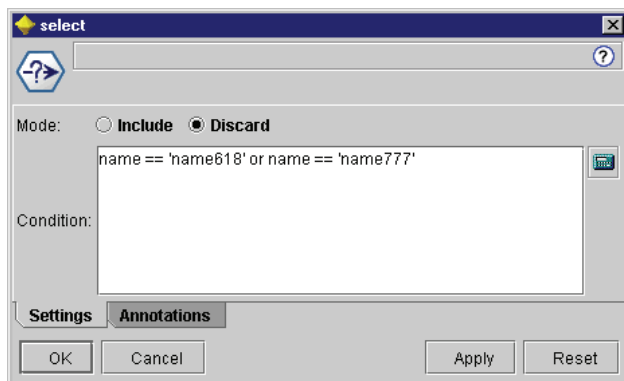
Figure 14-9
Distribution of grant applications



Value	Proportion	%	Count
name618		1.33	4
name777		0.67	2
name900		0.33	1
name899		0.33	1
name898		0.33	1
name897		0.33	1
name896		0.33	1
name895		0.33	1
name894		0.33	1
name893		0.33	1
name892		0.33	1
name891		0.33	1
name890		0.33	1
name889		0.33	1
name888		0.33	1
name887		0.33	1
name886		0.33	1
name885		0.33	1
name884		0.33	1
name883		0.33	1

Based on this, you can use a Select node to discard records for those farms that made multiple records.

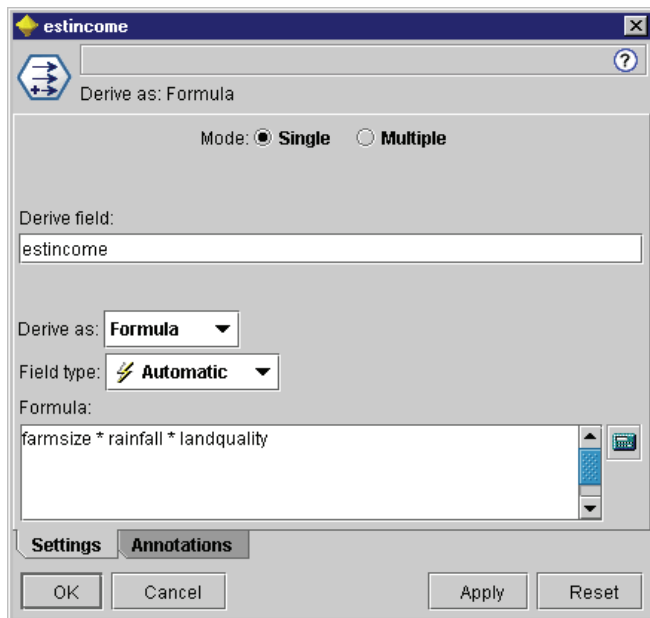
Figure 14-10
Discarding multiple claims



Next, you can focus on the characteristics of a single farm applying for aid. 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.

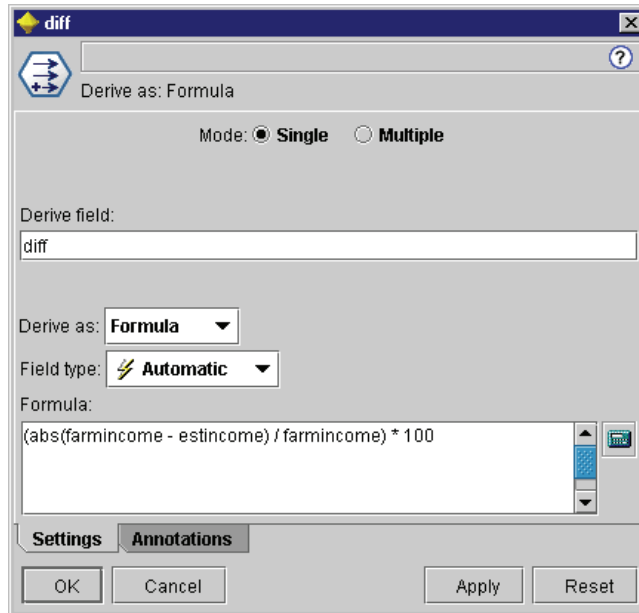
For example you can estimate income using a simple formula that multiplies farmsize * rainfall * landquality.

Figure 14-11
Estimating farm income



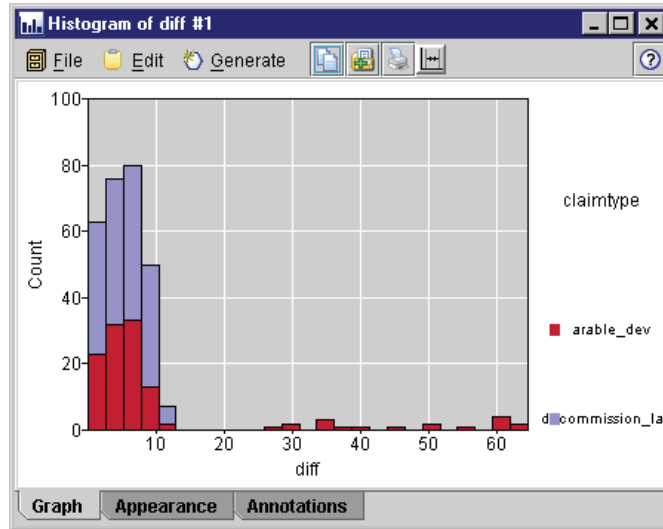
To investigate those farmers who deviate from the estimate, you need to derive another field that compares the two values and returns a percentage difference; this field will be called *diff*.

Figure 14-12
Comparing income differences



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.

Figure 14-13
Histogram of percentage difference



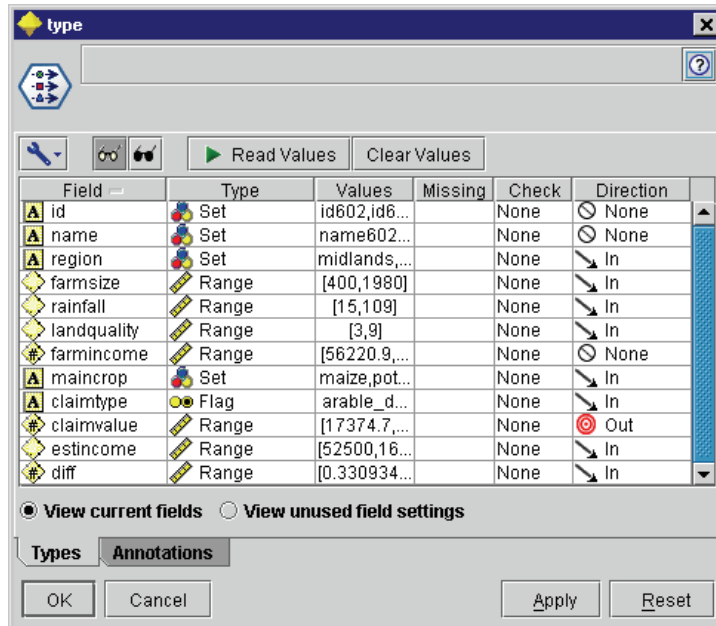
Since all of the large deviations seem to occur for *arable_dev* grants, it makes sense to select only *arable_dev* grant applications. To do so, attach a Select node to the Derive node called *diff* and select records using the CLEM expression `claimtype == 'arable_dev'`.

Training a Neural Network

From the initial data exploration, it is 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 *claimvalue* to *Out*.

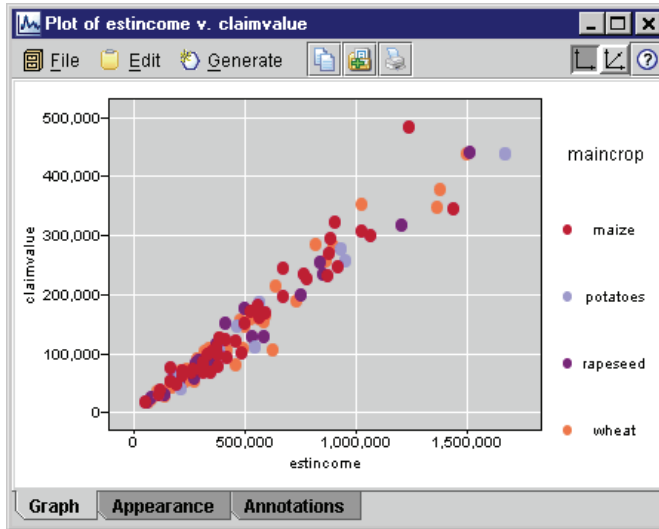
Figure 14-14
Input and Output variables for the neural network



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.

Figure 14-15

Comparing predicted and actual claim values



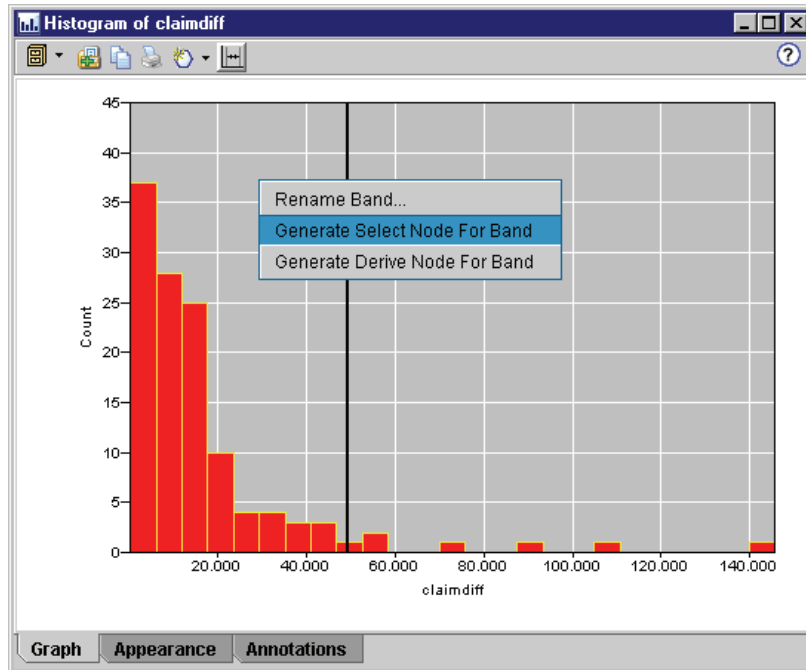
The fit appears to be good for the majority of cases. Derive another *claimdiff* field, similar to the “income differences” field derived earlier. This Derive node uses the CLEM expression

```
(abs(claimvalue - '$N-claimvalue') / 'claimvalue') * 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).

Figure 14-16

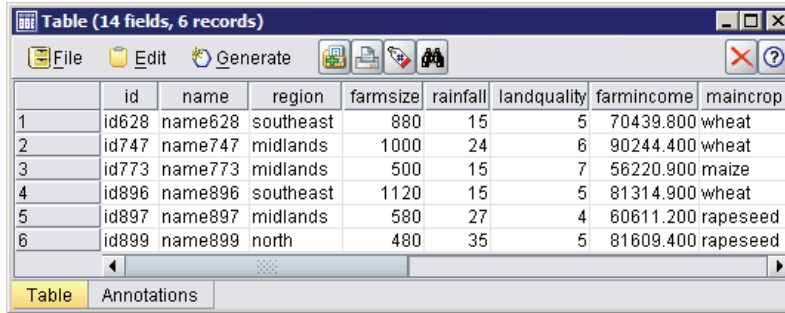
Selecting a subset of data from the histogram



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.

Figure 14-17

Records with unusually large claimdiff values



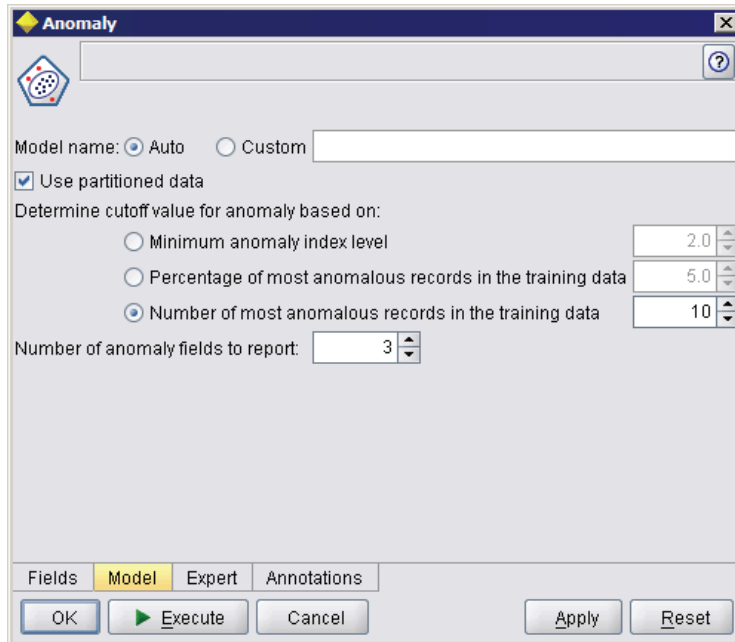
	id	name	region	farmsize	rainfall	landquality	farmincome	maincrop
1	id628	name628	southeast	880	15	5	70439.800	wheat
2	id747	name747	midlands	1000	24	6	90244.400	wheat
3	id773	name773	midlands	500	15	7	56220.900	maize
4	id896	name896	southeast	1120	15	5	81314.900	wheat
5	id897	name897	midlands	580	27	4	60611.200	rapeseed
6	id899	name899	north	480	35	5	81609.400	rapeseed

Anomaly Detection Revisited

As an alternative to using a Neural Net, you can use Anomaly Detection again, but this time including only the subset of records used in the Neural Net model (`claimtype == 'arable_dev'`).

Attach the Anomaly Detection node at the same point as the Neural Net node (so that both branch off of the same Type node). On the Model tab, select Number of most anomalous records in the training data as before, and enter 10 as the value.

Figure 14-18
Anomaly model settings



The image shows a software dialog box titled "Anomaly". It has a standard Windows-style title bar with a close button. Inside the dialog, there is a "Model name:" section with two radio buttons: "Auto" (selected) and "Custom" (with an empty text field). Below this is a checked checkbox labeled "Use partitioned data". The "Determine cutoff value for anomaly based on:" section contains three radio buttons: "Minimum anomaly index level" (with a value of 2.0), "Percentage of most anomalous records in the training data" (with a value of 5.0), and "Number of most anomalous records in the training data" (selected, with a value of 10). At the bottom of this section is a "Number of anomaly fields to report:" label with a value of 3. The dialog has four tabs at the bottom: "Fields", "Model" (highlighted in yellow), "Expert", and "Annotations". At the very bottom are five buttons: "OK", "Execute" (with a green play icon), "Cancel", "Apply", and "Reset".

Anomaly

Model name: ☒ Auto ☐ Custom

☒ Use partitioned data

Determine cutoff value for anomaly based on:


☐ Minimum anomaly index level

☐ Percentage of most anomalous records in the training data

☒ Number of most anomalous records in the training data

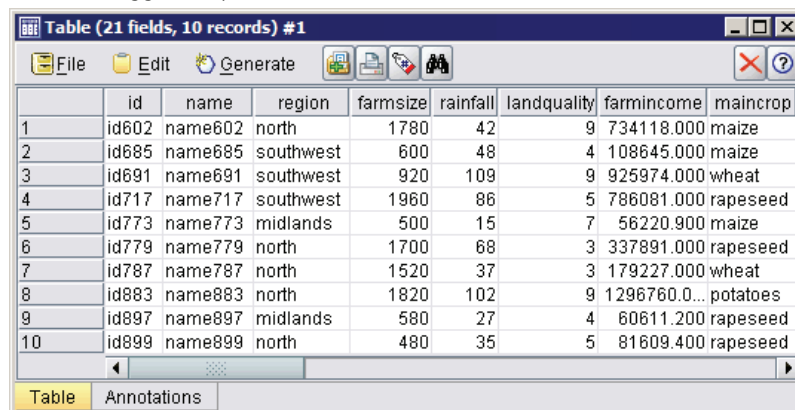
Number of anomaly fields to report:

Fields **Model** Expert Annotations

OK  Execute Cancel Apply Reset

Execute the node, add the generated model to the stream, and select the option to discard non-anomalous records as before. Attach a Table node and execute to see the result. The list of flagged records includes some of the same records as those selected by the Neural Net method (IDs 773, 897, and 899), while others are different.

Figure 14-19
Records flagged as potential anomalies



	id	name	region	farmsize	rainfall	landquality	farmincome	maincrop
1	id602	name602	north	1780	42	9	734118.000	maize
2	id685	name685	southwest	600	48	4	108645.000	maize
3	id691	name691	southwest	920	109	9	925974.000	wheat
4	id717	name717	southwest	1960	86	5	786081.000	rapeseed
5	id773	name773	midlands	500	15	7	56220.900	maize
6	id779	name779	north	1700	68	3	337891.000	rapeseed
7	id787	name787	north	1520	37	3	179227.000	wheat
8	id883	name883	north	1820	102	9	1296760.0...	potatoes
9	id897	name897	midlands	580	27	4	60611.200	rapeseed
10	id899	name899	north	480	35	5	81609.400	rapeseed

Keep in mind that these are exploratory methods and that different approaches may return different results. The best way to test the validity of either approach is to examine the records in question and see which turn out to be genuinely of interest (or fraudulent claims in this instance).

Summary

This example demonstrated two approaches for screening out possible cases of fraud—Anomaly Detection and a modeling approach based on a Neural Net.

After a preliminary screening using Anomaly Detection, you 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 relationship 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.

For purposes of comparison, the Anomaly Detection node was again used, but this time only on the arable development grants included in the Neural Net analysis. This approach gave results that were similar to the Neural Net method, but with some differences. This is to be expected, given that both are exploratory methods.

Market Basket Analysis (Rule Induction/C5.0)

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 *BASKETSIn*, 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 15-1
bask stream



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*

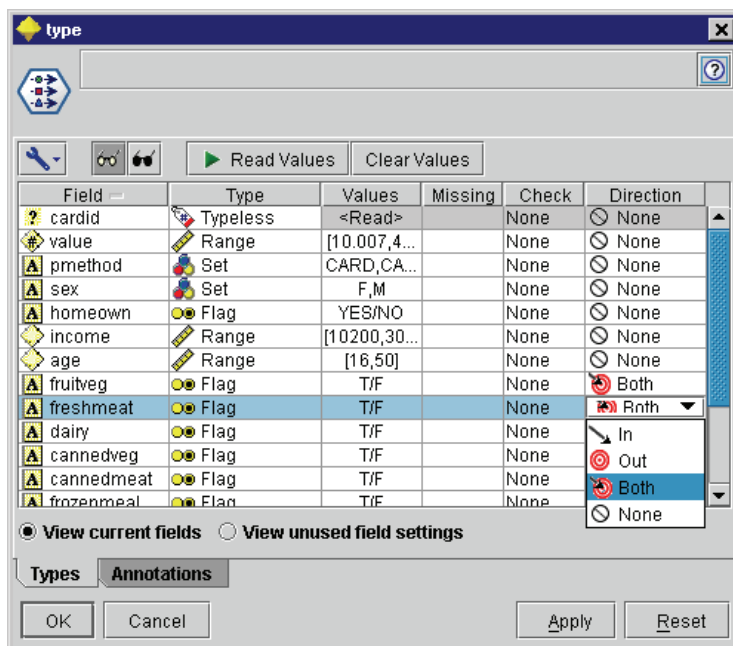
- *fish*
- *confectionery*

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 15-2
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 by using the context menu and selecting Browse.

Figure 15-3
Association rules

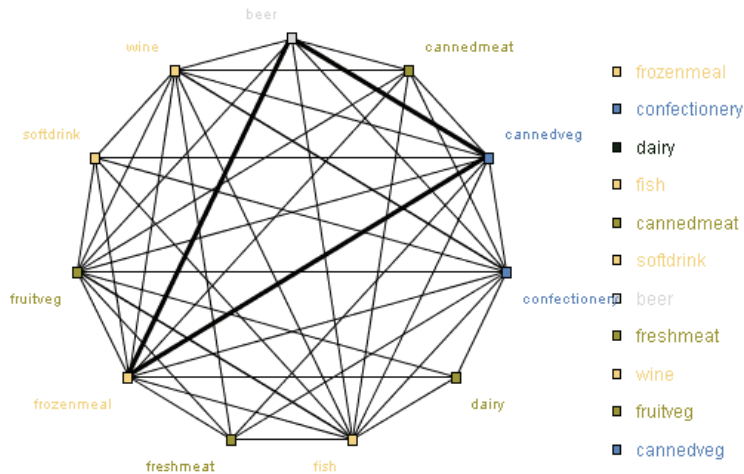
Consequent	Antecedent 1	Antecedent 2	Antecedent 3
frozenmeal	beer		
beer	cannedveg		
frozenmeal	cannedveg		
wine	confectionery		
beer	frozenmeal		
cannedveg	frozenmeal		
confectionery	wine		
cannedveg	cannedmeat	beer	
frozenmeal	cannedveg	beer	
cannedveg	frozenmeal	beer	
beer	cannedveg	frozenmeal	
frozenmeal	dairy	cannedveg	beer
frozenmeal	freshmeat	cannedveg	beer
frozenmeal	fruitveg	cannedveg	beer
cannedveg	cannedmeat	frozenmeal	beer
cannedveg	freshmeat	frozenmeal	beer
cannedveg	fruitveg	frozenmeal	beer
beer	dairy	cannedveg	frozenmeal
beer	freshmeat	cannedveg	frozenmeal
beer	fruitveg	cannedveg	frozenmeal

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 <= beer
beer <= 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 15-4
Web display of product associations

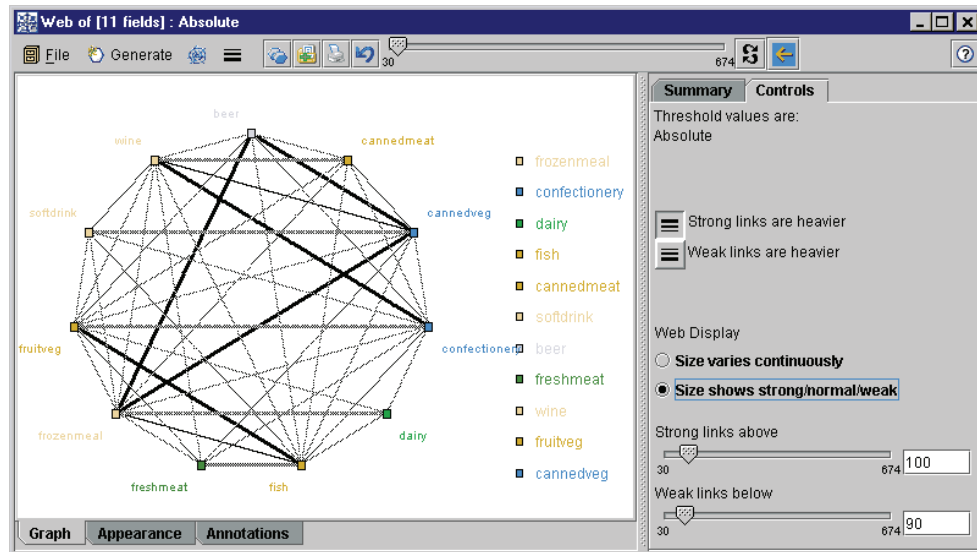


Because most combinations of product categories occur in several baskets, the strong links on this web are too numerous to show the groups of customers suggested by the GRI model. You need to raise the thresholds used by the web to show only the strongest links. To select these options, use the following 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 15-5
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”)

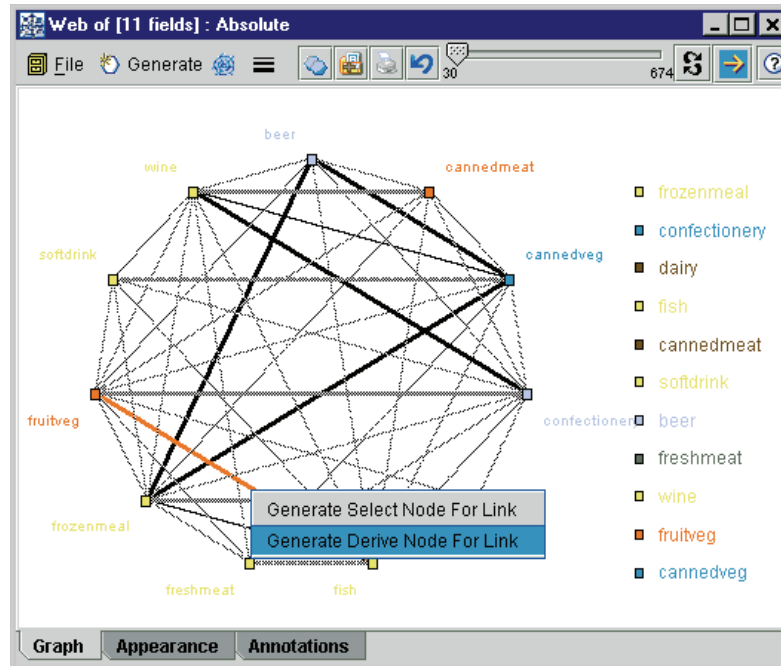
Note that GRI identified only the last two of these groups; the healthy eaters did not form a strong enough pattern for GRI to find it. The file *basklinks.str* contains the stream constructed so far.

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 15-6
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*.

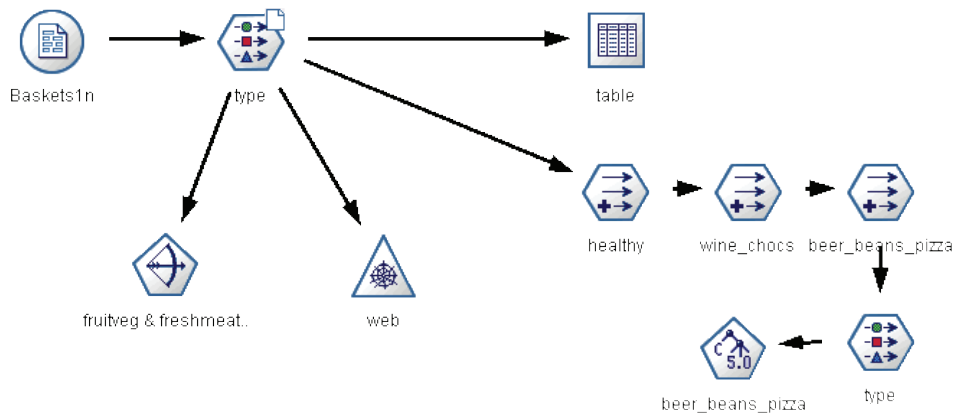
To profile these customer groups, connect the existing Type node to these three Derive nodes, and then attach another Type node. In the new Type node, set all fields to direction *None*, except for *value*, *pmethod*, *sex*, *homeown*, *income*, and *age*, which

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 15-7
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.

Retail Sales Promotion (Neural Net/C&RT)

This example deals with data that describes retail product lines and the effects of promotion on sales. (This data is 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 the exploration, data preparation, training, and test phases.

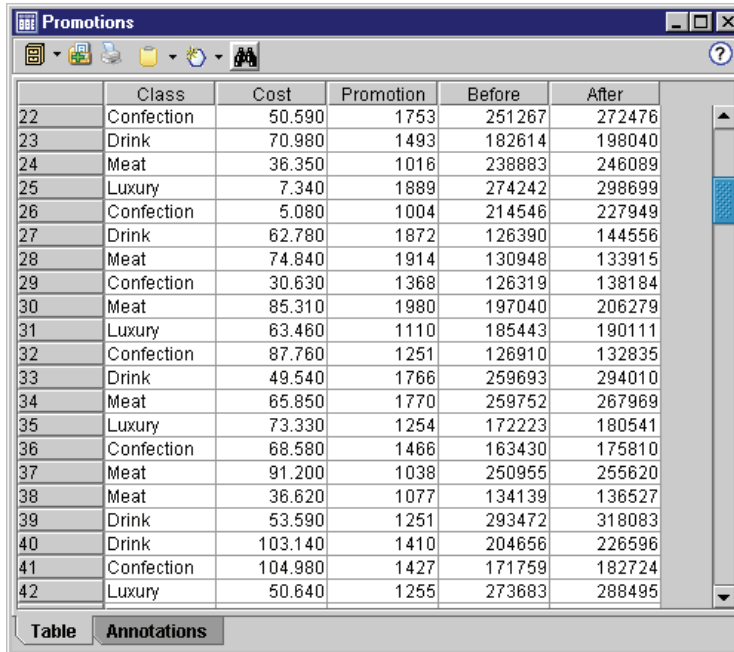
Examining the Data

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 following table:

Figure 16-1
Effects of promotion on product sales

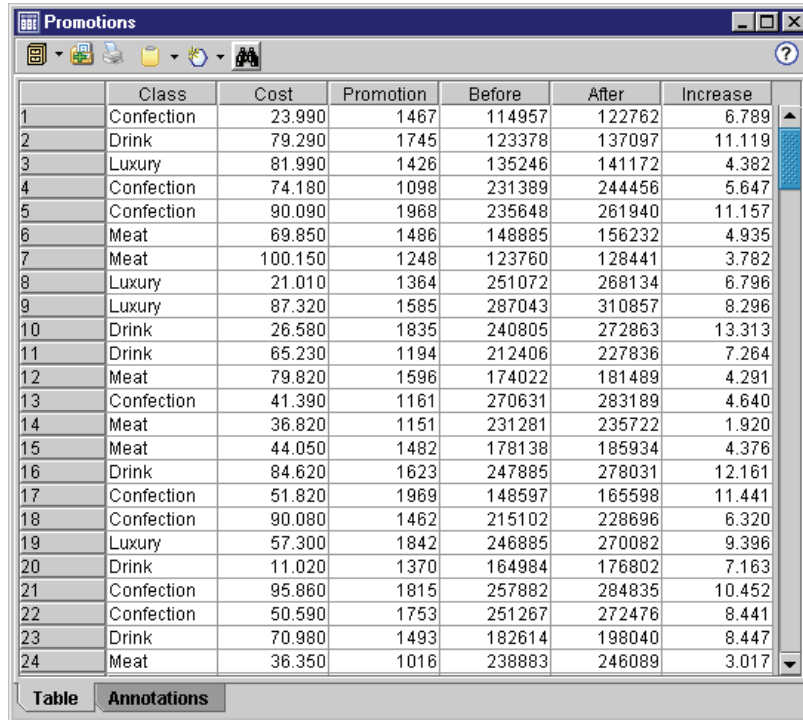


	Class	Cost	Promotion	Before	After
22	Confection	50.590	1753	251267	272476
23	Drink	70.980	1493	182614	198040
24	Meat	36.350	1016	238883	246089
25	Luxury	7.340	1889	274242	298699
26	Confection	5.080	1004	214546	227949
27	Drink	62.780	1872	126390	144556
28	Meat	74.840	1914	130948	133915
29	Confection	30.630	1368	126319	138184
30	Meat	85.310	1980	197040	206279
31	Luxury	63.460	1110	185443	190111
32	Confection	87.760	1251	126910	132835
33	Drink	49.540	1766	259693	294010
34	Meat	65.850	1770	259752	267969
35	Luxury	73.330	1254	172223	180541
36	Confection	68.580	1466	163430	175810
37	Meat	91.200	1038	250955	255620
38	Meat	36.620	1077	134139	136527
39	Drink	53.590	1251	293472	318083
40	Drink	103.140	1410	204656	226596
41	Confection	104.980	1427	171759	182724
42	Luxury	50.640	1255	273683	288495

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.

Figure 16-2
Increase in revenue after promotion

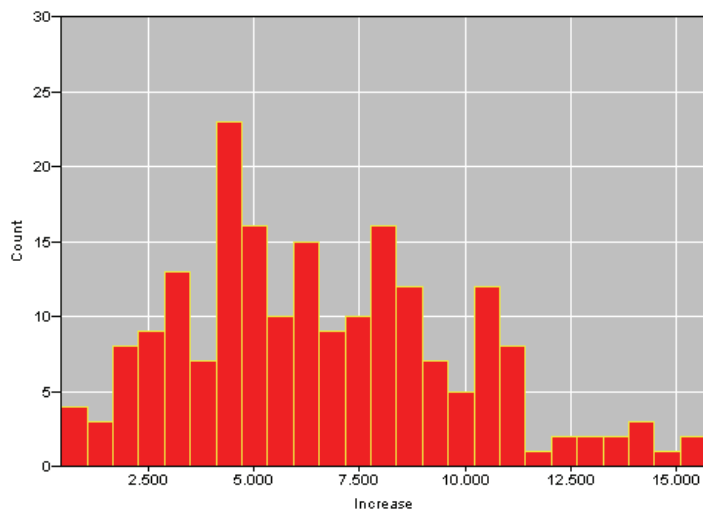


	Class	Cost	Promotion	Before	After	Increase
1	Confection	23.990	1467	114957	122762	6.789
2	Drink	79.290	1745	123378	137097	11.119
3	Luxury	81.990	1426	135246	141172	4.382
4	Confection	74.180	1098	231389	244456	5.647
5	Confection	90.090	1968	235648	261940	11.157
6	Meat	69.850	1486	148885	156232	4.935
7	Meat	100.150	1248	123760	128441	3.782
8	Luxury	21.010	1364	251072	268134	6.796
9	Luxury	87.320	1585	287043	310857	8.296
10	Drink	26.580	1835	240805	272863	13.313
11	Drink	65.230	1194	212406	227836	7.264
12	Meat	79.820	1596	174022	181489	4.291
13	Confection	41.390	1161	270631	283189	4.640
14	Meat	36.820	1151	231281	235722	1.920
15	Meat	44.050	1482	178138	185934	4.376
16	Drink	84.620	1623	247885	278031	12.161
17	Confection	51.820	1969	148597	165598	11.441
18	Confection	90.080	1462	215102	228696	6.320
19	Luxury	57.300	1842	246885	270082	9.396
20	Drink	11.020	1370	164984	176802	7.163
21	Confection	95.860	1815	257882	284835	10.452
22	Confection	50.590	1753	251267	272476	8.441
23	Drink	70.980	1493	182614	198040	8.447
24	Meat	36.350	1016	238883	246089	3.017

Table Annotations

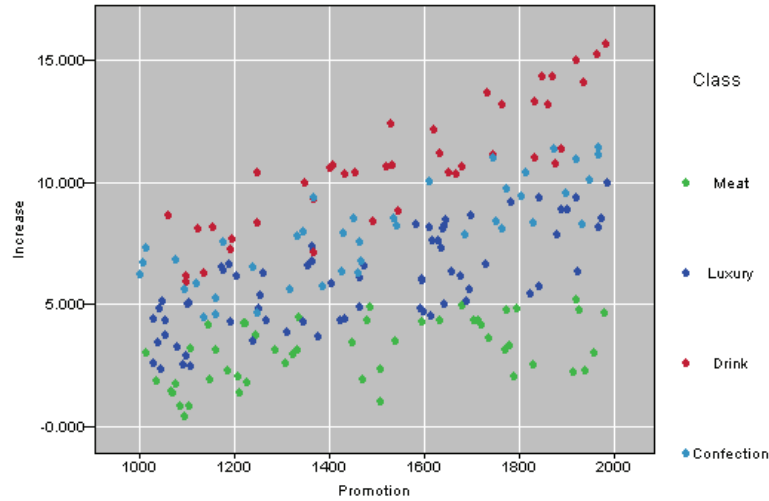
In addition, the stream displays a histogram of the increase and a scatterplot of the increase against the promotion costs expended, overlaid with the category of product involved.

Figure 16-3
Histogram of increase in revenue



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 available fields.

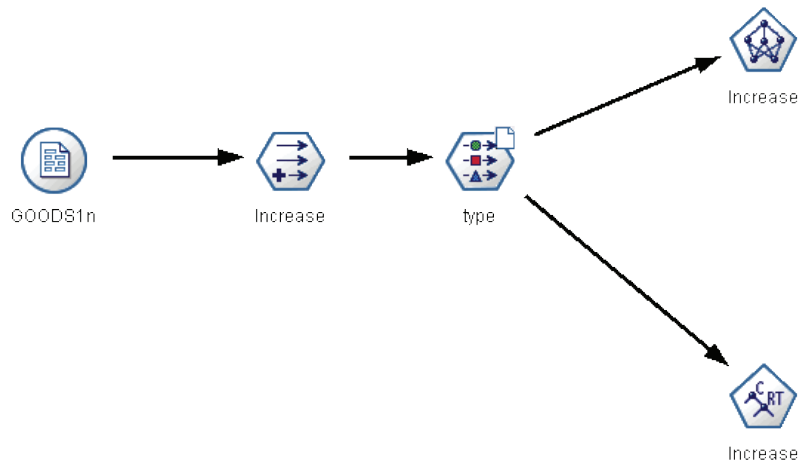
Figure 16-4
Revenue increase versus promotional expenditure



Learning and Testing

The stream `goodslearn.str` trains a neural network and a decision tree to make this prediction of revenue increase.

Figure 16-5
Modeling stream goodslearn.str



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.

Condition Monitoring (Neural Net/C5.0)

This example concerns monitoring status information from a machine and the problem of recognizing and predicting fault states. The data consists of a number of concatenated time series. Each record is a “snapshot” report on the machine in terms of the following:

- *Time*. An integer.
- *Power*. An integer.
- *Temperature*. 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:

Time	Power	Temperature	Pressure	Uptime	Status	Outcome
0	1059	259	0	404	0	0
1	1059	259	0	404	0	0
...						
51	1059	259	0	404	0	0
52	1059	259	0	404	0	0
53	1007	259	0	404	0	303
54	998	259	0	404	0	303

Time	Power	Temperature	Pressure	Uptime	Status	Outcome
			...			
89	839	259	0	404	0	303
90	834	259	0	404	303	303
0	965	251	0	209	0	0
1	965	251	0	209	0	0
			...			
51	965	251	0	209	0	0
52	965	251	0	209	0	0
53	938	251	0	209	0	101
54	936	251	0	209	0	101
			...			
208	644	251	0	209	0	101
209	640	251	0	209	101	101

This data, created using a simulation, is fictitious.

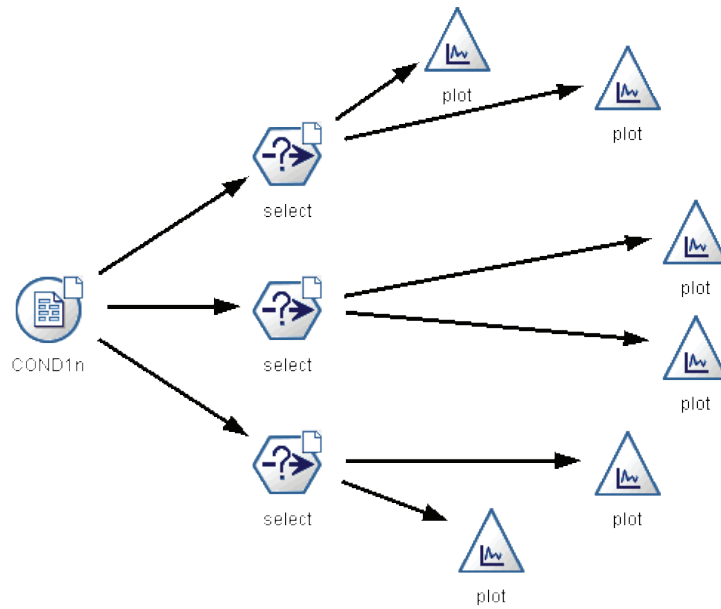
The following process is common to most data mining projects:

- 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

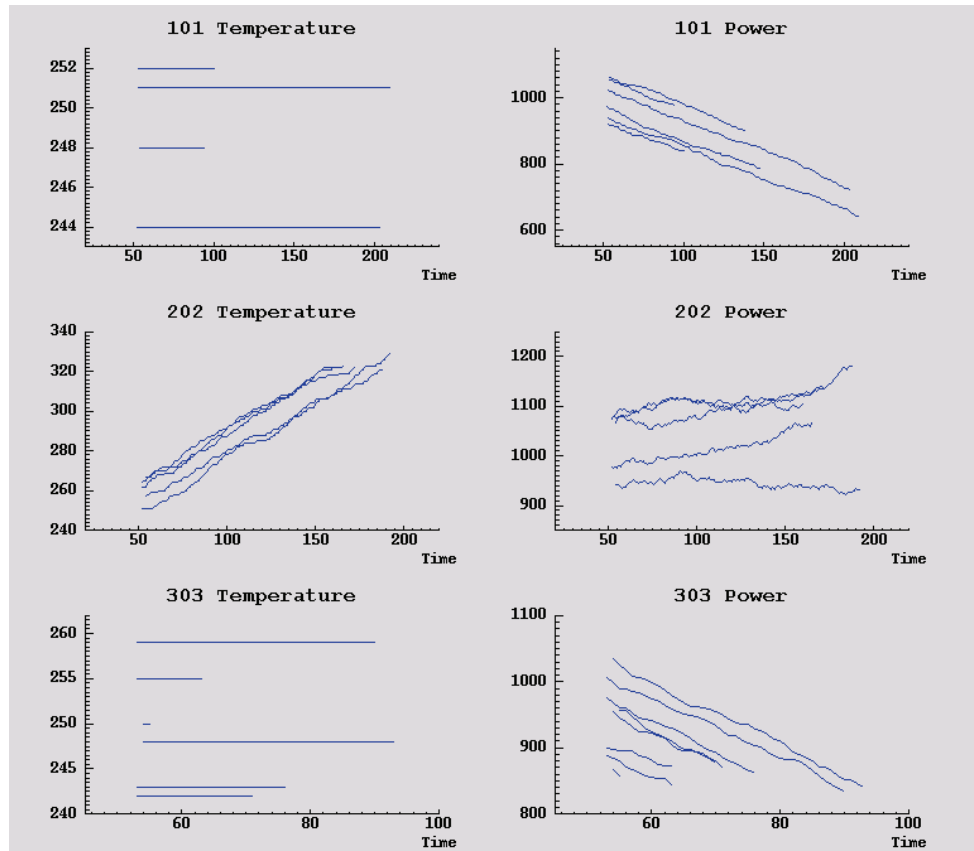
The file *condplot.str* illustrates the first part of the process. It contains the stream shown below, which plots a number of graphs. If the time series of temperature or power contains visible patterns, you could differentiate between impending error conditions or possibly predict their occurrence. 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 17-1
condplot stream



The results of this stream are shown in the following figure.

Figure 17-2
Temperature and power over time



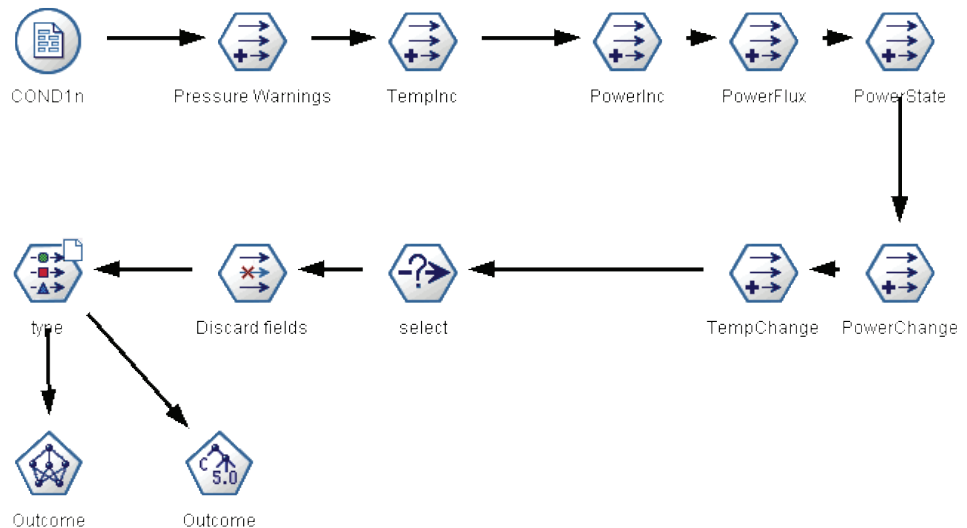
The graphs clearly display patterns distinguishing 202 errors from 101 and 303 errors. The 202 errors show rising temperature and fluctuating power over time; the other errors do not. However, patterns distinguishing 101 from 303 errors are less clear. Both errors show even temperature and a drop in 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 17-3
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 17-4
Models manager with generated nodes

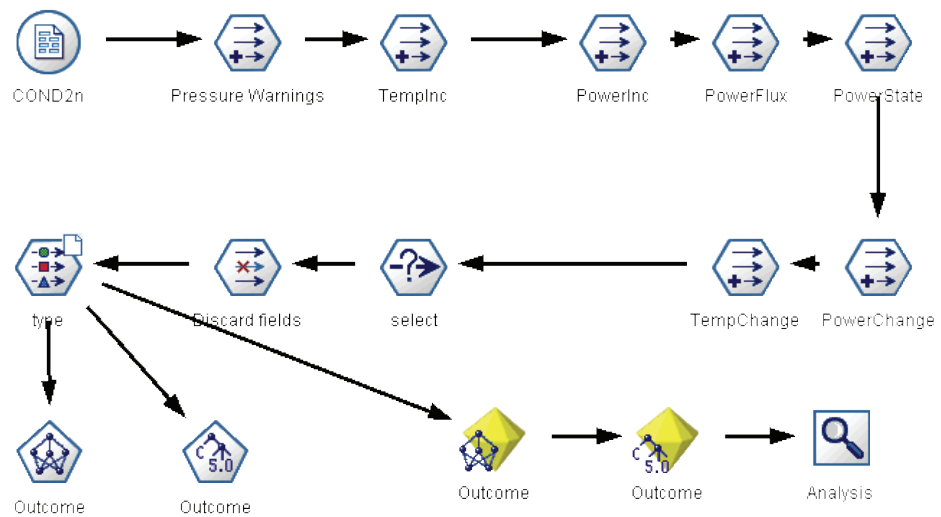


You can add generated model nodes to the existing stream to test the system or export the results of the model. In this example, we will test the results of the model.

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 17-5
Testing the trained network



Executing the Analysis node yields figures reflecting the accuracy of the trained network and rule.

Accessibility in Clementine

Overview of Clementine Accessibility

This release of Clementine offers greatly enhanced accessibility for all users, as well as specific support for users with visual and other functional impairments. This section describes the features and methods of working in Clementine using accessibility enhancements, such as screen readers and keyboard shortcuts.

Types of Accessibility Support

Whether you have a visual impairment or are dependent on the keyboard for manipulation, there is a wide variety of alternative methods for using this data mining toolkit. For example, you can build streams, specify options, and read output, all without using the mouse. Available keyboard shortcuts are listed in the topics that follow. Additionally, Clementine provides extensive support for screen readers, such as JAWS for Windows. You can also optimize the color scheme used in Clementine to provide additional contrast. These types of support are discussed in the following topics.

Accessibility for the Visually Impaired

There are a number of properties you can specify in Clementine that will enhance your ability to use the software.

Display Options

You can select colors for the display of graphs in Clementine. You can also choose to use your specific Windows settings for the software itself. This may help to increase visual contrast.

- ▶ To set display options, from the Tools menu, choose User Options.
- ▶ Then click the Display tab. The options on this tab include the software color scheme, chart colors, and font sizes for nodes.

Use of Sounds for Notification

By turning on or off sounds, you can control the way you are alerted to particular operations in the software. For example, you can activate sounds for events such as node creation and deletion or the generation of new output or models.

- ▶ To set notification options, from the Tools menu, choose User Options.
- ▶ Then click the Notifications tab.

Controlling the Automatic Launching of New Windows

The Notifications tab on the User Options dialog box is also used to control whether newly generated output, such as tables and charts, are launched in a separate window. It may be easier for you to disable this option and open an output window only when desired.

- ▶ To set these options, from the Tools menu, choose User Options.
- ▶ Then click the Notifications tab.
- ▶ In the dialog box, select New Output from the drop-down list on the right.
- ▶ Then, in the Open Window group, select Never.

Node Size

Nodes can be displayed in Clementine using either a standard or small size. You may want to adjust these sizes to fit your needs.

- ▶ To set node size options, from the File menu, choose Stream Properties.
- ▶ Then click the Layout tab.
- ▶ From the Icon Size drop-down list, select Standard.

Accessibility for Blind Users

Support for blind users is predominately dependent on the use of a screen reader, such as JAWS for Windows. To optimize the use of a screen reader with Clementine, you can specify a number of settings.

Display Options

Screen readers tend to perform better when the visual contrast is greater on the screen. If you already have a high-contrast Windows setting, you can choose to use these Windows settings for the software itself.

- ▶ To set display options, from the Tools menu, choose User Options.
- ▶ Then click the Display tab.

Use of Sounds for Notification

By turning on or off sounds, you can control the way you are alerted to particular operations in the software. For example, you can activate sounds for events such as node creation and deletion or the generation of new output or models.

- ▶ To set notification options, from the Tools menu, choose User Options.
- ▶ Then click the Notifications tab.

Controlling the Automatic Launching of New Windows

The Notifications tab on the User Options dialog box is also used to control whether newly generated output is launched in a separate window. It may be easier for you to disable this option and open an output window as needed.

- ▶ To set these options, from the Tools menu, choose User Options.
- ▶ Then click the Notifications tab.
- ▶ In the dialog box, select New Output from the drop-down list on the right.
- ▶ Then, in the Open Window group, select Never.

Keyboard Accessibility

Major new features have been added to make the product's functionality accessible from the keyboard. At the most basic level, you can press Alt plus the appropriate key to activate window menus (such as Alt-F to access the File menu) or press the Tab key to scroll through dialog box controls. However, there are special issues related to each of the product's main windows and helpful hints for navigating dialog boxes.

This section will cover the highlights of keyboard accessibility, from opening a stream to using node dialog boxes to working with output. Additionally, lists of keyboard shortcuts are provided for even more efficient navigation.

Shortcuts for Navigating the Main Window

You'll do most of your data mining work in the main window of Clementine. The main area is called the **stream canvas** and is used to build and run data streams. The bottom part of the window contains the **node palettes**, which contain all available nodes in Clementine. The palettes are organized on tabs corresponding to the type of data mining operation for each group of nodes. For example, nodes used to bring data into Clementine are grouped on the Sources tab, and nodes used to derive, filter, or type fields are grouped on the Field Ops tab (short for Field Operations).

The right side of the Clementine window contains several tools for managing streams, output, and projects. The top half on the right contains the **managers** and has three tabs that are used to manage streams, output, and generated models. You can access these objects by selecting the tab and an object from the list. The bottom half on the right contains the **project tool**, which allows you to organize your work into projects. There are two tabs in this area reflecting two different views of a project. The **Classes view** sorts project objects by type, while the **CRISP-DM view** sorts objects by the relevant data mining phase, such as Data Preparation or Modeling. These various aspects of the Clementine window are discussed throughout the Clementine Help system and User's Guide. For more information, see "Clementine Interface" in Chapter 3 on p. 13.

Following is a table of shortcuts used to move within the main Clementine window and build streams. Shortcuts for dialog boxes and output are listed in the topics that follow. Note that these shortcut keys are available only from the main window.

Main Window Shortcuts

Shortcut Key	Function
Ctrl-Tab	Moves focus between the main areas of the Clementine window.
Ctrl-F5	Moves focus to the node palettes.
Ctrl-F6	Moves focus to the stream canvas.
Ctrl-F7	Moves focus to the managers window.
Ctrl-F8	Moves focus to the projects window.

Node and Stream Shortcuts

Shortcut Key	Function
Ctrl-N	Creates a new blank stream canvas.
Ctrl-O	Opens the Open Stream dialog box.
Ctrl-number keys	Moves focus to the corresponding tab. For example, within a tabbed pane or window, Ctrl-1 moves to the first tab starting from the left, Ctrl-2 to the second, etc.
Ctrl-down arrow	Used in the node palette to move from the tabs to the tab contents, such as History node or Database node.
Ctrl-up arrow	Used in the node palette to move from the tab contents back up to the tabs for node groups, such as Source or Record Ops.
Enter	When a node is selected in the palettes (including refined models in the generated models palette), this keystroke adds the node to the stream canvas. Clicking Enter on a node already in the canvas opens that node's dialog box.
Ctrl-Enter	When a node is selected in the palette, adds that node to the stream canvas without selecting it and while keeping focus on the node palettes.
Alt-Enter	When a node is selected in the palette, adds that node to the stream canvas and selects that node while also keeping focus on the node palettes. This method is useful for building a continuous data stream of nodes in the stream canvas.
Space	When a node has focus, selects the node.
Shift-space	When a node has focus, allows for multiple selection.
Arrow	Cycles between tabs or moves between items in the node palette.
Alt-Arrow	Moves selected nodes on the stream canvas in the direction of the arrow keys.
Ctrl-A	Selects all nodes.

Shortcut Key	Function
Ctrl-Q	When a node has focus, selects it and all nodes downstream.
Ctrl-W	Toggles with Ctrl-Q.
Ctrl-Alt-D	Duplicate node.
Ctrl-Alt-L	Load node.
Ctrl-Alt-R	Rename node.
Ctrl-Alt-U	Create User Input node.
Ctrl-Alt-C	Toggle the cache for a node on/off.
Ctrl-Alt-F	Flush cache.
Tab	Cycles through all base nodes on the current stream canvas.
Any key	In the current stream, gives focus and cycles to the next node whose name starts with the key pressed.
F2	Starts the connection process for a node selected in the canvas. Use the Tab key to move to the desired node on the canvas, and press the spacebar to finish the connection.
F3	Destroys all connections for the selected node on the canvas.
Delete	Delete selected node or selected connection.
Backspace	Delete selected node or selected connection.
Shift-F10	Opens context menu.
Ctrl-Alt-X	Expand SuperNode.
Ctrl-Alt-Z	Zoom in / zoom out of a SuperNode.
Ctrl-E	With focus in the stream canvas, this executes the current stream.

A number of standard shortcut keys are also used in Clementine, such as Ctrl-C to copy. For more information, see “Using Shortcut Keys” in Chapter 3 on p. 17.

Shortcuts for Dialog Boxes and Tables

Several shortcut and screen reader keys are helpful when you are working with dialog boxes, tables, and tables in dialog boxes. A complete list of special keyboard and screen reader shortcuts is provided below.

Dialog and Expression Builder Shortcuts

Shortcut Key	Function
Alt-4	Used to dismiss all open dialog boxes or output windows. Output can be retrieved from the Output tab in the managers window.
Ctrl-End	With focus on any control in the Expression Builder, this will move the insertion point to the end of the expression.
Ctrl-1	In the Expression Builder, moves focus to the expression edit control.
Ctrl-2	In the Expression Builder, moves focus to the function list.
Ctrl-3	In the Expression Builder, moves focus to the field list.

Table Shortcuts

Table shortcuts are used for output tables as well as table controls in dialog boxes for nodes such as Type, Filter, and Merge. Typically, you will use the Tab key to move between table cells and Ctrl-Tab to leave the table control. *Note:* Occasionally, a screen reader may not immediately begin reading the contents of a cell. Pressing the arrows keys once or twice will reset the software and start the speech.

Shortcut Key	Function
Ctrl-W	For tables, reads the short description of the selected row. For example, "Selected row 2 values are sex, flag, m/f, etc."
Ctrl-Alt-W	For tables, reads the long description of the selected row. For example, "Selected row 2 values are field = sex, type = flag, sex = m/f, etc."
Ctrl-D	For tables, reads the short Description of the selected area. For example, "Selection is one row by six columns."
Ctrl-Alt-D	For tables, provides the long Description of the selected area. For example, "Selection is one row by six columns. Selected columns are Field, Type, Missing. Selected row is 1."
Ctrl-T	For tables, provides a short description of the selected columns. For example, "Fields, Type, Missing."
Ctrl-Alt-T	For tables, provides a long description of the selected columns. For example, "Selected columns are Fields, Type, Missing."
Ctrl-R	For tables, provides the number of Records in the table.
Ctrl-Alt-R	For tables, provides the number of Records in the table as well as column names.

Shortcut Key	Function
Ctrl-I	For tables, reads the cell I nformation, or contents, for the cell that has focus.
Ctrl-Alt-I	For tables, reads the long description of cell I nformation (column name and contents of the cell) for the cell that has focus.
Ctrl-G	For tables, provides short G eneral selection information.
Ctrl-Alt-G	For tables, provides long G eneral selection information.
Ctrl-Q	For tables, provides a Q uick toggle of the table cells. Ctrl-Q reads long descriptions, such as “Sex=Female,” as you move through the table using the arrow keys. Selecting Ctrl-Q again will toggle to short descriptions (cell contents).

Shortcut Keys Example: Building Streams

In order to make the stream-building process more clear for users dependent on the keyboard or on a screen reader, following is an example of building a stream without the use of the mouse. In this example, you will build a stream containing a Variable File node, a Derive node, and a Histogram node using the following steps:

- ▶ **Start Clementine.** When Clementine first starts, focus is in the stream canvas.
- ▶ **Ctrl+Tab.** Moves to the Streams tab of the managers.
- ▶ **Tab 4 times.** Moves to the Favorites tab of the node palettes.
- ▶ **Ctrl+down arrow.** Moves focus from the tab itself to the body of the tab.
- ▶ **Right arrow.** Moves focus to the Variable File node.
- ▶ **Spacebar.** Selects the Variable File node.
- ▶ **Ctrl+Enter.** Adds the Variable File node to the stream canvas but keeps focus on the node palette. This way, you are ready to move to the next node and add it without moving back and forth between the canvas and the palette. This key combination also keeps selection on the Variable File node so that the next node added will be connected to it.
- ▶ **Right arrow 4 times.** Moves to the Derive node.
- ▶ **Spacebar.** Selects the Derive node.

- ▶ **Alt+Enter.** Adds the Derive node to the canvas and moves selection to the Derive node. This node is now ready to be connected to the next added node.
- ▶ **Right arrow 5 times.** Moves focus to the Histogram node in the palette.
- ▶ **Spacebar.** Selects the Histogram node.
- ▶ **Enter.** Adds the node to the stream and moves focus to the stream canvas.

To proceed to the next example on editing nodes, [click here](#).

Shortcut Keys Example: Editing Nodes

In this example, you will use the stream built in the earlier example. The stream consists of a Variable File node, a Derive node, and a Histogram node. The instructions begin with focus on the third node in the stream, the Histogram node.

- ▶ **Ctrl+ left arrow 2 times.** Moves focus back to the Variable File node.
- ▶ **Enter.** Opens the Variable File dialog box. Tab through to select the desired file. Then close the dialog box.
- ▶ **Ctrl+right arrow.** Gives focus to the second node, a Derive node.
- ▶ **Enter.** Opens the Derive node dialog box. Tab through to select fields and specify derive conditions. Then close the dialog box.
- ▶ **Ctrl+right arrow.** Gives focus to the third node, a Histogram node.
- ▶ **Enter.** Opens the Histogram node dialog box. Tab through to select fields and specify graph options. Then close the dialog box.

At this point, you can add additional nodes or execute the current stream. Keep in mind the following tips when you are building streams:

- When manually connecting nodes, use F2 to create the start and end points of a connection, and use the spacebar to finalize the connection.
- Use F3 to destroy all connections for a selected node in the canvas.
- Once you have created a stream, use Ctrl-E to execute the current stream.

A complete list of shortcut keys is available. For more information, see “Shortcuts for Navigating the Main Window” on p. 282.

Using a Screen Reader

A number of screen readers are available on the market. Clementine has been tested with JAWS for Windows.

Due to the nature of Clementine's unique graphical representation of the data mining process, charts and graphs are optimally used visually. It is possible, however, for you to understand and make decisions based on output and models viewed textually using a screen reader.

Using the Clementine Dictionary File

A Clementine dictionary file (*Awt.JDF*) is available for inclusion with JAWS. To use this file:

- ▶ Navigate to the */accessibility* subdirectory of your Clementine installation and copy the dictionary file (*Awt.JDF*).
- ▶ Copy it to the directory with your JAWS scripts.

You may already have a file named *Awt.JDF* on your machine if you have other JAVA applications running. In this case, you may not be able to use this dictionary file without manually editing the dictionary file.

Alternate Help System for Screen Readers

An alternate version of the Clementine Help system is available to provide additional support for screen readers. To switch to the alternate version, you will need to run a batch file to convert the existing Help system into a format that is more efficiently read by a screen reader.

To run the batch file:

- ▶ Navigate to the root directory of the Help system where the file *help.js* resides. Use the directory where you installed Clementine and navigate down to *help/i18n/* and the folder for your language or locale. For English-speaking users, the file path is: *help/i18n/English_US/*.
- ▶ To switch to the screen-reader-friendly format, run the batch file *format-for-screen-readers.bat*.

- To switch back to the default format, run the batch file *restore-default-format.bat*.

Using a Screen Reader with HTML Output

When viewing output displayed as HTML within Clementine using a screen reader, you may encounter some difficulties. The following types of output are affected:

- Output viewed on the Advanced tab for Regression, Logistic Regression, and Factor/PCA nodes
- Report node output

In each of these windows or dialog boxes, there is a tool on the toolbar that can be used to launch the output into your default browser, which provides standard screen reader support. You can then use the screen reader to convey the output information.

Accessibility in the Tree Builder

The standard display of a decision tree model in the Tree Builder may cause problems for screen readers. To access an accessible version, from the Tree Builder menus choose:

View
Accessible Window

This displays a view similar to the standard tree map, but one which Jaws can read correctly. You can move up, down, right, or left using the standard arrow keys. As you navigate the accessible window, the focus in the Tree Builder window moves accordingly. Use the space bar to change the selection, or use Ctrl-Space to extend the current selection.

Tips for Use

There are several tips for making the Clementine environment more accessible to you. The following are general hints when working in Clementine.

- **Exiting extended text boxes.** Use Ctrl-Tab to exit extended text boxes. *Note:* Ctrl-Tab is also used to exit table controls.

- **Using the Tab key rather than arrow keys.** When selecting options for a dialog box, use the Tab key to move between option buttons. The arrow keys will not work in this context.
- **Drop-down lists.** In a drop-down list for dialog boxes, you can use either the Escape button or spacebar to select an item and then close the drop-down list. You can also use the Escape key to close drop-down lists that do not close when you have tabbed to another control.
- **Execution status.** When you are executing a stream on a large database, JAWS can lag behind in reading the execution status to you. Press the Ctrl key periodically to update the status reporting.
- **Using the node palettes.** When you first enter a tab of the node palettes, JAWS will sometimes read “groupbox” instead of the name of the node. In this case, you can use Ctrl+right arrow and then Ctrl+left arrow to reset the screen reader and hear the node name.
- **Reading menus.** Occasionally, when you are first opening a menu, JAWS may not read the first menu item. If you suspect that this may have happened, use the down arrow and then the up arrow to hear the first item in the menu.
- **Cascaded menus.** JAWS does not read the first level of a cascaded menu. If you hear a break in speaking while moving through a menu, press the right arrow key to hear the child menu items.

Additional tips for use are discussed at length in the following topics.

Interference with Other Software

When testing Clementine with screen readers, such as JAWS, our development team discovered that the use of a Systems Management Server (SMS) within your organization may interfere with JAWS’ ability to read Java-based applications, such as Clementine. Disabling SMS will correct this situation. Visit the Microsoft Web site for more information on SMS.

JAWS and Java

Different versions of JAWS provide varying levels of support for Java-based software applications. Although Clementine will work with all recent versions of JAWS, some versions may have minor problems when used with Java-based systems. Visit the JAWS for Windows Web site at <http://www.FreedomScientific.com>.

Using Graphs in Clementine

Visual displays of information, such as histograms, evaluation charts, multiplots, and scatterplots, are difficult to interpret with a screen reader. Please note, however, that web graphs and distributions can be viewed using the textual summary available from the output window.

Unicode Support

Unicode Support in Clementine

Clementine 10.0 is fully Unicode-enabled for both the Server and Client. This makes it possible to exchange data with other applications that support Unicode, including multi-language databases, without any loss of information that might be caused by conversion to or from a locale-specific encoding scheme.

- Clementine stores Unicode data internally and can read and write multi-language data stored as Unicode in databases without loss.
- Clementine can read and write UTF-8 encoded text files. Text file import and export will default to the locale-encoding but support UTF-8 as an alternative. This setting can be specified in the file import and export nodes, or the default encoding can be changed in the Stream Properties dialog box. For more information, see “Setting Options for Streams” in Chapter 5 on p. 77.
- SPSS, SAS, and text data files stored in the locale-encoding will be converted to UTF-8 on import and back again on export. When writing to any file, if there are Unicode characters that do not exist in the locale character set, they will be substituted and a warning will be displayed. This should occur only where the data has been imported from a data source that supports Unicode (a database or UTF-8 text file) and that contains characters from a different locale or from multiple locales or character sets.
- Clementine Solution Publisher images are UTF-8 encoded and are truly portable between platforms and locales.

About Unicode

The goal of the Unicode standard is to provide a consistent way to encode multilingual text so that it can be easily shared across borders, locales, and applications. The Unicode Standard, now at version 4.0.1, defines a character set that is a superset of all

of the character sets in common use in the world today and assigns to each character a unique name and code point. The characters and their code points are identical to those of the Universal Character Set (UCS) defined by ISO-10646. For more information, see the Unicode Home Page (<http://www.unicode.org>).

Features Added in Previous Releases

New Features in Clementine 9.0

This release of Clementine adds support for interactive tree building along with two new tree-building algorithms, a new Partition node, Unicode support, and improved performance and scalability. In addition, new integrations with SPSS Model Manager, Oracle Data Mining, and IBM Intelligent Miner help to leverage the power of Clementine in building enterprise-level applications, and a new release of Text Mining for Clementine allows you to gain greater value from data mining by incorporating unstructured text data into your analyses.

New Features

Interactive tree building. Build decision trees interactively using C&R Tree, CHAID, and QUEST nodes. You can allow the algorithm to choose the best split at each level or to customize splits as needed, applying your business knowledge to refine or simplify the tree.

- Two new tree-building algorithms are provided: CHAID, or **Chi-squared Automatic Interaction Detection**, is a classification method for building decision trees by using chi-square statistics to identify optimal splits. Exhaustive CHAID, a modification that does a more thorough job of examining all possible splits, is also supported. QUEST, or **Quick, Unbiased, Efficient Statistical Tree**, is an alternative method that computes quickly and avoids the other methods' biases in favor of predictors with many categories.
- Edit and prune the tree level-by-level, and access gains, risks, and related information before generating one or more models.

- Use a partition field to separate the data into training and test samples (see below).
- Import projects from AnswerTree 3.0 into Clementine.

Data partitioning. The new Partition node automatically derives a field that splits the data into separate subsets, or samples, for the training, testing, and validation stages of model building. Alternatively, an existing field can be designated to use as a partition. By using one sample to generate the model and a separate sample to test it, you can get a good indication of how well the model will generalize to larger data sets that are similar to the current data.

In-database modeling and scoring. New database integrations support modeling with Oracle Data Mining and IBM Intelligent Miner, allowing customers to leverage the power of their databases using native algorithms provided by these vendors. These features can be accessed through Clementine's easy-to-use graphical user interface. By combining the analytical capabilities and ease-of-use of the Clementine desktop with the power and performance of a database, you get the best of both worlds.

Model management. Clementine 9.0 supports the publishing and sharing of objects through SPSS Model Manager, including streams, models, and output files (*.cou), allowing you to manage the life cycle of data mining models and related predictive objects. Models are stored in a central repository where they can be shared with other researchers and applications and tracked using extended versioning, metadata, and search capabilities. For more information, see "Predictive Enterprise Repository" in Chapter 11 on p. 203.

Performance and scalability. Improved performance and scalability through global memory and disk cache allocation allows resources to be managed more effectively at the system level. This results in fewer calls to the memory manager and a reduction in the size and number of disk caches. Sort performance has also been improved.

Unicode support. Clementine is fully Unicode-enabled, making it possible to exchange data with other applications that support Unicode, including multi-language databases, without any loss of information that might be caused by conversion to or from a locale-specific encoding scheme. For more information, see "Unicode Support in Clementine" in Appendix B on p. 293.

Text mining. This release adds support for Text Mining for Clementine 3.0, an add-on product that uses linguistic methods to extract key concepts from text based on context. The new release of Text Mining is fully integrated with Clementine and can be deployed with applications such as PredictiveCallCenter to support real-time scoring of unstructured data. To streamline the modeling process, the new Concept Extraction

Browser allows you to select and merge extracted concepts directly into your data. The new release also includes Text Mining Builder, which allows you to fine-tune the extraction process through an easy-to use graphical environment.

Additional Enhancements

Extended support for SQL optimization. Clementine can generate SQL for many operations that can be “pushed back” to the database for execution, resulting in improved performance on the client side. For this release, SQL generation has been added for neural network and logistic regression models. A preview pane, allowing you to view generated SQL, has also been added.

Extended SmartScore support. SmartScore is the SPSS model scoring technology, based on the PMML standard for describing data mining models. Previously, SmartScore was used to score TwoStep and Association models and other models that have been imported from PMML files. For Clementine 9.0, SmartScore will also be used to score C&RT, CHAID, and QUEST decision tree models and logistic regression models, ensuring that scoring results will be consistent if deployed across other SPSS tools.

Metadata support. Support for metadata used by SPSS has been extended, including missing value ranges, date formats, and number display formats (such as currency and scientific notation). This allows for a more seamless transfer of data between Clementine and SPSS without losing metadata.

Output to file object. Tables, graphs, and reports generated from Clementine output nodes can now be saved in output object (*.cou) format. This can also be specified in scripting by setting the output_format property to **Object**.

Explicit control over output names. This release includes the ability to specify the name of an output object (graph, table, report, etc.) so that the name of the output can be specified before it is constructed.

Scripting enhancements include the following:

- The ability to include blocks of literal text by wrapping them in triple quotes (""") so that embedded quotes, spaces, and returns are recognized as such
- Script control over the default type property behavior (*Read* versus *Pass*)

New Features in Release 8.5

The following is a comprehensive list of enhancements and changes incorporated in Clementine 8.5. For information on current features, see “New Features in Clementine 10.0” in Chapter 2 on p. 3.

Major New Features

- New CARMA model.
- Direct scoring of Apriori and CARMA models.
- Support for in-database modeling with Microsoft Decision Trees.
- Support for field and value labels.
- Field-level formatting options.
- Support for ordinal metadata.
- New export options with the Predictive Applications Wizard.
- Visualization enhancements for association models, including the ability to use Intelligent Miner Visualization directly from Clementine.

Additional Enhancements

- Easy access to Clementine Application Templates (CATs).
- Clementine 64-bit support on Solaris 9.
- Scoring of TwoStep using SmartScore with new PMML user options.
- Memory management options for building Neural Network, K-Means, Kohonen, and Apriori models.
- Addition of Wald and Score statistics to the Stepwise method of Logistic Regression.
- Ability to specify a custom CREATE TABLE statement when exporting from Database and Publisher nodes.
- Type and Filter functionality now available from all source nodes.
- Ability to run Clementine Server as a non-root process. For more information, see the *Clementine Server Administrator's Guide*.
- Scripting changes designed to increase options available for automation and promote consistency in naming and usage.

New Features in Release 8.0

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

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.
- 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.
- Smart navigation using Browse dialog boxes.

Glossary

This glossary defines terms used in Clementine and data mining in general.

Glossary A-C

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 is then weighted to emphasize the records for which the

first model generated errors and the second model is built. The data is 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.

CHAID. Chi-squared Automatic Interaction Detection, a classification method for building decision trees by using chi-squared statistics to identify optimal splits.

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 cross-tabulation.

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-DM 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.

Glossary D-F

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 has 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. This data has been cleaned and organized for easy access. Often includes a metadata store as well.

decile. A division of data into 10 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 has 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.

Glossary G-I

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.

Glossary J-M

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 cross-tabulation 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.

Glossary N-Q

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 has 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 is 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.

projects 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.

QUEST. Quick, Unbiased, Efficient Statistical Tree is a classification method for building binary decision trees. This method computes quickly and avoids the other methods' biases in favor of predictors with many categories.

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.

Glossary R-S

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 that you analyze is based on a sample; the conclusions that 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.

Glossary T-Z

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.

Unicode. A standard for encoding multilingual text data so it can easily be shared across borders, locales, and applications.

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.

- 508 compliance, 279
- abs function, 141
- accessibility, 279, 291
 - example, 286–287
 - features in Clementine, 279
 - Help system, 288
 - tips in Clementine, 289
- adding
 - to a project, 169
- aggregating data, 301
- algorithms, 40
- allbutfirst function, 145
- allbutlast function, 145
- alphabefore function, 145
- and operator, 140
- annotating, 301
 - nodes, 74
 - streams, 74, 88
- annotations
 - folder, 175
 - project, 174
- Anomaly Detection node, 36
- antecedent, 301
- applications, 34
- applications of data mining, 50
- Apriori, 301
- arccos function, 142
- arccosh function, 142
- arcsin function, 142
- arsinh function, 142
- arctan function, 142
- arctan2 function, 142
- arctanh function, 142
- association, 301
- association rules, 47
- attribute, 33
- automation, 111
- backup stream files
 - restoring, 90
- balance, 301
- banding continuous variables, 41
- batch mode, 32, 111, 301
 - invoking software, 10
- bitwise functions, 143
- @BLANK function, 108, 137, 162
- blank handling
 - CLEM functions, 162
- blanks, 103, 105, 118, 301
- Boolean field, 301
- boosting, 301
- build rule node
 - loading, 92
- business understanding, 301
- C&R Trees, 40, 301
- C5.0, 40, 301
- cache, 301
 - enabling, 21, 71
 - flushing, 73, 77
 - options for nodes, 71
 - saving, 73
 - setting up a cache, 70

- cache file node
 - loading, 92
- case, 33, 301
- category merging, 41
- cdf_chisq function, 143
- cdf_f function, 143
- cdf_normal function, 143
- cdf_t function, 143
- cell, 301
- CEMI, 301
- CHAID, 40, 302
- characters, 129, 131
- charts
 - saving output, 92
- checking CLEM expressions, 127
- chi-square distribution
 - probability functions, 143
- classes, 165, 168
- classification, 301
- classification and regression trees, 301
- classification trees, 301
- CLEM, 122, 301
 - building expressions, 124
 - checking expressions, 127
 - datatypes, 130–132
 - examples, 111
 - expressions, 115, 129
 - functions, 124
 - introduction, 32, 111
 - language, 129
- CLEM expressions
 - parameters, 84, 116
- CLEM functions
 - bitwise, 143
 - blanks and nulls, 162
 - comparison, 139
 - conversion, 138
 - datetime, 151
 - global, 161
 - information, 137
 - list of available, 135
 - logical, 140
 - missing values, 108
 - numeric, 141
 - probability, 143
 - random, 145
 - sequence, 155, 157
 - special functions, 163
 - string, 145
 - trigonometric, 142
- Clementine, 12
 - accessibility features, 279
 - documentation, 7
 - features at a glance, 13
 - getting started, 9
 - options, 20
 - overview, 3, 9, 295
 - running from command line, 10
 - tips and shortcuts, 100
- Clementine Application Templates (CATs)
 - data mapping tool, 94
 - opening, 57
 - using as examples, 57
- Clementine Batch, 2
- Clementine Client, 1
- Clementine Server, 1
- Clementine Solution Publisher, 1
- Cleo, 185
 - stream prerequisites, 197
 - wizard, 196, 198
- client
 - default directory, 22
- clustering, 46, 301

- colors
 - setting, 26
- comma, 77
- command line
 - starting Clementine, 10
- comparison functions, 139
- concatenating strings, 138
- condition monitoring, 271
- conditions, 115
- confidence, 301
- connections, 301
 - server, 10
- consequent, 301
- continuous variables
 - segmenting, 41
- conventions, 136
- conversion functions, 138
- copy, 15
- correlations, 301
- cos function, 142
- cosh function, 142
- count_equal function, 121, 139
- count_greater_than function, 121, 139
- count_less_than function, 121, 139
- count_not_equal function, 121, 139
- count_nulls function, 108, 121, 139
- count_substring function, 145
- CRISP-DM, 165
 - projects view, 166
- CRISP-DM process model, 52–53, 301
- cross-tabulation, 301
- cross-validation, 301
- currency display format, 79
- customer_dbase.sav* file, 225
- cut, 15
- data
 - availability, 50
 - balancing, 56
 - coverage, 51
 - expertise, 52
 - reduction, 41
 - size of, 51
- data audit node
 - use in data mining, 51
 - use in exploration, 33
- data cleaning, 303
- data mapping tool, 94–95
- data mining, 33, 303
 - application examples, 57
 - choosing a modeling method, 55
 - strategy, 52
 - tips, 55
- data preparation, 303
- data quality, 303
- data set, 303
- data streams
 - building, 61–62
- data understanding, 303
- data visualization, 303
- data warehouse, 303
- datatypes, 114
 - in parameters, 87
- date formats, 78, 132–133
- date functions, 132–133
 - date_before, 139, 151
 - date_days_difference, 151
 - date_in_days, 151
 - date_in_months, 151
 - date_in_weeks, 151
 - date_in_years, 151
 - date_months_difference, 151
 - date_weeks_difference, 151

- date_years_difference, 151
 - @TODAY function, 151
- date_before function, 139
- date/time values, 120
- datetime functions
 - datetime_date, 151
 - datetime_day, 151
 - datetime_day_name, 151
 - datetime_day_short_name, 151
 - datetime_hour, 151
 - datetime_in_seconds, 151
 - datetime_minute, 151
 - datetime_month, 151
 - datetime_month_name, 151
 - datetime_month_short_name, 151
 - datetime_now datetime_second, 151
 - datetime_time, 151
 - datetime_timestamp, 151
 - datetime_weekday, 151
 - datetime_year, 151
- decile, 303
- decimal places
 - display formats, 79
- decimal symbol, 77
 - number display formats, 79
- decision tree models, 38
- decision trees, 303
 - accessibility, 289
- default
 - project phase, 167
- degrees
 - measurements units, 78
- delimiters, 303
- denormalized data, 303
- dependent variable, 303
- deployment, 303
- derived field, 303
- diagram, 303
- dictionary file, 288
 - @DIFF function, 155, 157
- directed web, 303
- direction of fields, 303
- directory
 - default, 22
- display formats
 - currency, 79
 - decimal places, 79
 - grouping symbol, 79
 - numbers, 79
 - scientific, 79
- distribution, 303
- distribution functions, 143
- div function, 141
- documentation, 7
- DTD, 199–200
- encoding, 79, 293
- endstring function, 145
- equals operator, 139
- equation, 303
- error messages, 82
- essential fields, 94, 98
- evaluation, 303
- examples
 - Clementine Application Templates (CATs), 57
 - condition monitoring, 271
 - fraud detection, 233
 - market basket analysis, 255
 - overview, 223
 - retail analysis, 265
- exceptions, 57
- execution
 - for streams, 89
- Exhaustive CHAID, 40

- exponential function, 141
- exporting
 - PMML, 199, 202
 - to Cleo, 196
- Expression Builder, 284
 - accessing, 123
 - overview, 122
 - using, 124
- expressions, 129
- f* distribution
 - probability functions, 143
- factor, 289
- factor analysis, 303
- feature, 303
- Feature Selection models, 225
- Feature Selection node
 - importance, 35, 56, 225
 - missing values, 107
 - ranking predictors, 35, 56, 225
 - screening predictors, 35, 56, 225
- featureselection.str* file, 225
- field, 303
 - @FIELD function, 108, 163
- fields, 33, 129, 132
 - in CLEM expressions, 125
 - ranking importance, 35, 56, 225
 - screening, 35, 56, 225
 - selecting for analysis, 35, 56, 225
 - viewing values, 126
- @FIELDS_BETWEEN function, 108, 121, 163
- @FIELDS_MATCHING function, 108, 121, 163
- filler node, 303
 - missing values, 108
- filtering fields, 303
- fixed files, 303
- flags, 303
- flat file, 303
- fonts, 26
- fracof function, 141
- fraud detection, 233
 - Anomaly Detection, 36
- functions, 132–133, 136–137, 155, 157
 - @BLANK, 108
 - examples, 111
 - @FIELD, 122, 163
 - @GLOBAL_MAX, 161
 - @GLOBAL_MEAN, 161
 - @GLOBAL_MIN, 161
 - @GLOBAL_SDEV, 161
 - @GLOBAL_SUM, 161
 - handling missing values, 108
 - in CLEM expressions, 124
 - @PARTITION, 163
 - @PREDICTED, 122, 163
 - @TARGET, 122, 163
- generated models, 306
- global functions, 161
- global values, 306
- graphs
 - adding to projects, 168
 - saving output, 92
- greater than operator, 139
- GRI node, 306
- grouping symbol
 - number display formats, 79
- hasendstring function, 145
- hasmidstring function, 145
- hasstartstring function, 145
- hassubstring function, 145

- Help, 288
 - accessing, 19
 - types of, 19
- hints
 - general usage, 100
- histogram, 306
- history, 306
- hot keys, 17
- HTML output
 - screen reader, 289
- icons
 - setting options, 80
- if, then, else functions, 140
- importance
 - ranking predictors, 35, 56, 225
- importing
 - PMML, 200, 202
- impurity, 306
- @INDEX function, 155, 157
- information functions, 137
- input field, 306
- instantiation, 306
- integer_bitcount function, 143
- integer_leastbit function, 143
- integer_length function, 143
- integers, 129–130, 306
- interaction, 306
- interaction identification, 41
- interactive tree builder
 - accessibility, 289
- interactive trees, 38
- intof function, 141
- introduction, 129
 - Clementine, 9
- is_date function, 137
- is_datetime function, 137
- is_integer function, 137
- is_number function, 137
- is_real function, 137
- is_string function, 137
- is_time function, 137
- is_timestamp function, 137
- isalphacode function, 145
- isendstring function, 145
- islowercode function, 145
- ismidstring function, 145
- isnumbercode function, 145
- isstartstring function, 145
- issubstring function, 145
- issubstring_count function, 145
- issubstring_lim function, 145
- isuppercode function, 145
- iterative, 306
- Java, 291
- JAWS, 279, 288, 290–291
- k*-means clustering, 46, 307
- K-Means node
 - large sets, 77
- keyboard shortcuts, 282, 284
- keywords
 - annotating nodes, 74
- knowledge discovery, 33
- Kohonen networks, 43, 46, 307
- Kohonen node
 - large sets, 77
- labels
 - displaying, 80
 - value, 200
 - variable, 200

- language
 - options, 20
- @LAST_NON_BLANK function, 155, 157, 162
- length function, 145
- less than operator, 139
- lift, 307
- linear regression, 44, 307
 - export as PMML, 31
- lists, 129, 131
- loading
 - nodes, 92
 - states, 92
- locale
 - options, 20
- locchar function, 145
- locchar_back function, 145
- log files
 - displaying generated SQL, 29
- log function, 141
- log10 function, 141
- logical functions, 140
- logistic regression, 44, 289, 307
 - export as PMML, 31
- lowertoupper function, 145
- machine learning, 33, 307
- main effect, 307
- mandatory fields, 99
- mapping data, 98
- mapping fields, 94
- market basket analysis, 255, 307
- matches function, 145
- matrix, 307
- max function, 139
 - @MAX function, 155, 157
- max_n function, 121, 139
- mean, 307
 - MEAN function, 155, 157
 - @MEAN function, 155, 157
 - mean_n function, 121, 141
 - median, 307
 - member function, 139
- memory
 - managing, 20–21
- merging records, 307
- messages
 - displaying generated SQL, 29
- metadata, 52, 307
- middle mouse button
 - simulating, 17, 65
- min function, 139
 - @MIN function, 155, 157
- min_n function, 121, 139
- minimizing, 16
- misclassification matrix, 307
- missing values, 105, 107, 118
 - CLEM expressions, 108
 - filling, 103
 - handling, 103
 - in records, 106
- mod function, 141
- mode, 307
- modeling, 307
- models
 - adding to projects, 168
 - exporting, 30
- mouse
 - using in Clementine, 17, 65
- multilayer perceptrons, 42, 307
- multiplot node, 307
- naming nodes and streams, 74
- navigating
 - keyboard shortcuts, 282

- negate function, 141
- NetGenesis Web analytics technology, 2
- neural net node, 42
 - large sets, 77
- neural networks, 42, 308
- new features, 3, 295, 298–299
- node caching
 - enabling, 71
- node names, 74
- nodes, 9, 308
 - adding, 65, 68
 - adding to projects, 168–169
 - annotating, 74
 - bypassing in a stream, 67
 - connecting in a stream, 65
 - deleting, 65
 - deleting connections, 69
 - duplicating, 70
 - editing, 70
 - introduction, 63
 - loading, 92
 - saving, 90
 - setting options, 70
- noisy data, 51
- normal distribution
 - probability functions, 143
- normalized data, 308
- not equal operator, 139
- not operator, 140
- notifications
 - setting options, 23
- @NULL function, 108, 137, 162
- nulls, 103, 118
- number display formats, 79
- numbers, 119, 130
- numeric functions, 141
- objects
 - properties, 176
- ODBC, 308
- @OFFSET function, 155, 157
- oneof function, 145
- opening
 - models, 92
 - nodes, 92
 - output, 92
 - projects, 169
 - states, 92
 - streams, 92
- operator precedence, 134
- operators
 - joining strings, 138
- optimization
 - options, 27
 - parallel processing, 27
 - SQL generation, 27
- options, 20
 - display, 26
 - for Clementine, 20
 - optimization, 27
 - PMML, 30
 - stream properties, 77, 80, 82
 - user, 22
- options.cfg*, 27
- or operator, 140
- outliers, 308
 - identifying, 36
- output field, 308
- output files
 - saving, 92
- overfitting, 308
- palettes, 308
 - overview, 13

- parallel processing
 - enabling, 27
 - user options, 27
- parameters, 308
 - session, 84, 87, 116
 - stream, 84, 87, 116
 - type, 87
- @PARTITION_FIELD function, 163
- paste, 15
- PCA, 308
- performance
 - node caching, 71
- period, 77
- pi function, 142
- PMML
 - export options, 30
 - exporting models, 199, 202
 - importing models, 200, 202
- PMML models
 - linear regression, 31
 - logistic regression, 31
- power (exponential) function, 141
- PowerPoint files, 168
- precedence, 134
- @PREDICTED function, 163
- prediction, 41, 308
- Predictive Applications Wizard, 185–186, 189
 - stream prerequisites, 186
- Predictive Enterprise Repository, 185, 203, 210
 - adding and removing folders, 219
 - connecting to, 204
 - deleting objects, 215
 - deleting versions, 215
 - object properties, 212
 - permissions for folders, 219
 - permissions for objects, 212
 - properties of folders, 219
 - retrieving objects, 208
 - searching in, 216
 - storing objects, 206
 - transferring projects to, 171
- Predictive Framework, 185
- PredictiveMarketing, 185–186
- predictors, 308
 - ranking importance, 35, 56, 225
 - screening, 35, 56, 225
 - selecting for analysis, 35, 56, 225
- printing, 31
 - streams, 70
- probabilities
 - in predictive applications, 188
- probability, 308
- probability functions, 143
- projects, 165
 - adding objects, 169
 - annotating, 174
 - building, 169
 - Classes view, 168
 - closing, 176
 - creating new, 169
 - CRISP-DM view, 166
 - folder properties, 175
 - generating reports, 177
 - in Predictive Enterprise Repository, 171
 - object properties, 176
 - setting a default folder, 167
 - setting properties, 172
- projects tool, 308
- properties
 - for data streams, 77
 - project folder, 175
 - report phases, 177
- pruning, 308
- purple nodes, 27

- pushbacks, 27
- Quality node
 - missing values, 107
- quantile, 308
- quartile, 308
- query, 308
- QUEST, 40, 310
- quintile, 308
- radial basis function network (RBFN), 310
- radians
 - measurements units, 78
- random function, 145
- random0 function, 145
- ranking predictors, 35, 56, 225
- reals, 129–130, 310
- records, 33, 310
 - missing values, 106
- refined models, 310
- refresh
 - source nodes, 77
- regression, 289
- regression trees, 310
- relational database, 310
- rem function, 141
- renaming
 - nodes, 74
 - streams, 88
- replace function, 145
- replicate function, 145
- report manager, 310
- reports
 - adding to projects, 168
 - generating, 177
 - saving output, 92
 - setting properties, 177
- resizing, 16
- retail analysis, 265
- retrieving objects from Predictive Enterprise Repository, 208
- rollover days, 79
- rough diamond, 310
- round function, 141
- row, 310
- rule induction, 38, 310
- rulesets, 310
 - evaluating, 77
- sampling, 56, 310
- SAS files
 - encoding, 293
- saving
 - multiple objects, 91
 - nodes, 90
 - output objects, 92
 - states, 90–91
 - streams, 90
- scatterplots, 310
- scenario, 196
- scientific notation
 - display format, 79
- scoring, 310
- screen readers, 282, 284, 288–289
 - example, 286–287
- screening predictors, 35, 56, 225
- scripting, 32, 111, 310
- scrolling
 - setting options, 80
- @SDEV function, 155, 157
- sdev_n function, 121, 141
- searching for objects in Predictive Enterprise Repository, 216

- segmentation, 41, 310
- select, 310
- self-organizing maps, 43
- sensitivity analysis, 310
- sequence detection, 47
- sequence functions, 155, 157
- sequences, 310
- server
 - default directory, 22
- server mode, 10
- session parameters, 84, 87, 116
- set command, 84, 116
- set field type, 310
- sets, 77
- shortcuts
 - general usage, 100
 - keyboard, 17, 282, 284
- sign function, 141
- significance (statistical), 310
- sin function, 142
- @SINCE function, 155, 157
- sinh function, 142
- skipchar function, 145
- skipchar_back function, 145
- slot parameters, 310
- Solution Publisher, 1
- solutions template library, 94
- soundex function, 150
- soundex_difference function, 150
- source nodes
 - data mapping, 95
 - refreshing, 77
- spaces
 - removing from strings, 117, 145
- special characters
 - removing from strings, 117
- special functions, 163
- SPSS files
 - encoding, 293
- SQL, 310
- SQL generation, 27
 - logging, 29
 - previewing, 29
- sqrt function, 141
- standard deviation, 310
- startstring function, 145
- states
 - loading, 92
 - saving, 90–91
- statistical models, 44
- statistics, 310
- status window, 13
- stop execution, 15
- storing objects in Predictive Enterprise Repository, 206
- stratification, 41
- stream canvas
 - overview, 13
 - settings, 80
- stream names, 74
- stream parameters, 84, 87, 116
- stream rewriting
 - SQL optimization, 27
- streams, 9, 310
 - adding nodes, 65, 68
 - adding to projects, 168–169
 - annotating, 74, 88
 - backup files, 90
 - building, 61–62
 - bypassing nodes, 67
 - connecting nodes, 65
 - execution, 89
 - loading, 92
 - options, 77

- renaming, 74
- saving, 90
- string functions, 145
- strings, 129, 131, 310
 - manipulating in CLEM expressions, 117
 - matching, 117
 - replacing, 117
- stripchar function, 145
- strmember function, 145
- subscrs function, 145
- substring function, 145
- substring_between function, 145
- @SUM function, 155, 157
- sum_n function, 121, 141
- SuperNode
 - parameters, 84, 116
- supervised learning, 310
- support, 310
- symbolic field, 310
- system
 - options, 20
- system-missing values, 103
- t* distribution
 - probability functions, 143
- tables, 284
 - adding to projects, 168
 - saving output, 92
- tan function, 142
- tanh function, 142
- target, 313
- @TARGET function, 163
- temp directory, 12
- template fields, 99
- templates, 94–95, 196
- testbit function, 143
- @TESTING_PARTITION function, 163
- text data files
 - encoding, 293
- text encoding, 79
- Text Mining for Clementine, 2
- @THIS function, 155, 157
- throughput reporting, 27
- time and date functions, 132–133
- time formats, 78, 132–133
- time functions, 132–133
 - time_before, 139, 151
 - time_hours_difference, 151
 - time_in_hours, 151
 - time_in_mins, 151
 - time_in_secs, 151
 - time_mins_difference, 151
 - time_secs_difference, 151
- time series, 313
- time_before function, 139
- tips
 - for accessibility, 289
 - general usage, 100
- to_date function, 138
- to_datetime function, 138
- to_integer function, 138
- to_number function, 138
- to_real function, 138
- to_string function, 138
- to_time function, 138
- to_timestamp function, 138
- @TODAY function, 151
- toolbar, 15
- ToolTips
 - annotating nodes, 74
- @TRAINING_PARTITION function, 163
- transformations, 313
- tree builder
 - accessibility, 289

- tree-based analysis
 - general uses, 41
 - typical applications, 34
 - trigonometric functions, 142
 - trim function, 145
 - trim_start function, 145
 - trimend function, 145
 - TwoStep clustering, 46, 313
 - type, 313
 - Type node
 - missing values, 108
 - typical applications, 34
 - undef function, 162
 - undo, 15
 - Unicode, 313
 - Unicode support, 293
 - unmapping fields, 94
 - unrefined models, 47, 313
 - unsupervised learning, 46, 313
 - uppertolower function, 145
 - upstream, 313
 - user input node, 313
 - user options, 22
 - user-missing values, 103
 - UTF-8 encoding, 79, 293
 - visual programming, 12, 313
 - warnings, 82
 - setting options, 23
 - Web Mining for Clementine, 2
 - web node, 313
 - what's new, 3, 295, 298–299
 - whitespace
 - removing from strings, 117, 145
 - wizard
 - accessing, 196
 - overview, 196
 - zooming, 15
-
- @VALIDATION_PARTITION function, 163
 - values, 114
 - adding to CLEM expressions, 126
 - viewing from a data audit, 126
 - variable file node, 313
 - variables, 33, 313
 - screening, 41
 - variance, 313
 - vingtile, 313