

Automotive Warranty Data Analysis on the Worldwide Web

S. Antoniadou, E. Barker, L. Dodge, B. Reading, M. Senger, and S. Walters
Trillium Teamologies, Inc., Royal Oak, MI

ABSTRACT

Trillium Teamologies assembled a team possessing SAS and Java expertise to create a warranty analysis web application for an automotive OEM. This paper discusses development strategies considered in building the application, including architectural and security considerations. Criteria are provided for deciding among client load models that range from thin (webAF® model) through integrated object (Enterprise Guide® model) to full client (SAS/AF® model) approaches. This paper includes some of the development considerations and decisions made in web-enabling an enterprise wide application:

- choice of web technology (CGI v. Java)
- applet v. servlet methodology
- development environment (AppDevStudio® v. Forte)
- populating list box controls (on-demand access v. server page caching)
- processing large tasks through a batch queue facility
- using ODS to deliver reports in *.pdf form

Choice of model is dependent on cost, complexity, control, and response time requirements.

INTRODUCTION

The automotive industry spends billions of dollars to cover warranty claims each year. Analysis of repair claims is key to improvement in vehicle design, production, maintenance and image. Reliability is associated with warranty history and directly impacts the market value of new cars sold, lease residual values, and used car values. Quality, reliability, and durability indices are the data mining gems of warranty analysis. Automotive companies strive to improve these indicators, which impact both sales and profitability.

A leading automotive manufacturer had developed a warranty analysis system called QWIK (Quality With Information and Knowledge) as a SAS/AF client server application. This system provides quality control, reliability, and durability feedback to both internal (OEM) and supplier audiences. QWIK analysis serves as a basis for allocating financial incentives to suppliers who meet and exceed quality standards set by the OEM. Likewise, the analysis of this data provides serves as one basis for risk-sharing between the OEM and the supplier community.

The OEM selected Trillium Teamologies (TTI) to develop QWIK Web, a web version of the QWIK full client product. QWIK Web is designed to reduce client-side software licensing costs, simplify report production, and re-use existing code.

Now available are QWIK (as a fully featured client/server application), QWIK Web (as a thin client subset of QWIK), and Enterprise Guide (for freestyle reporting against the data). To meet the customer's requirements, the team was confronted with a number of decisions concerning methodology, design, and choice of development toolsets. This paper offers a simple set of criteria in helping make such decisions, and a discussion of how other business requirements were met.

SOFTWARE REQUIREMENTS

QWIK Web reflects a three-tier architecture: browser, web server, and application server. The browser tier requires Microsoft Internet Explorer 5.0 or Netscape 4.1 and Adobe Acrobat Reader 5.0. The web server tier requires Sun iPlanet

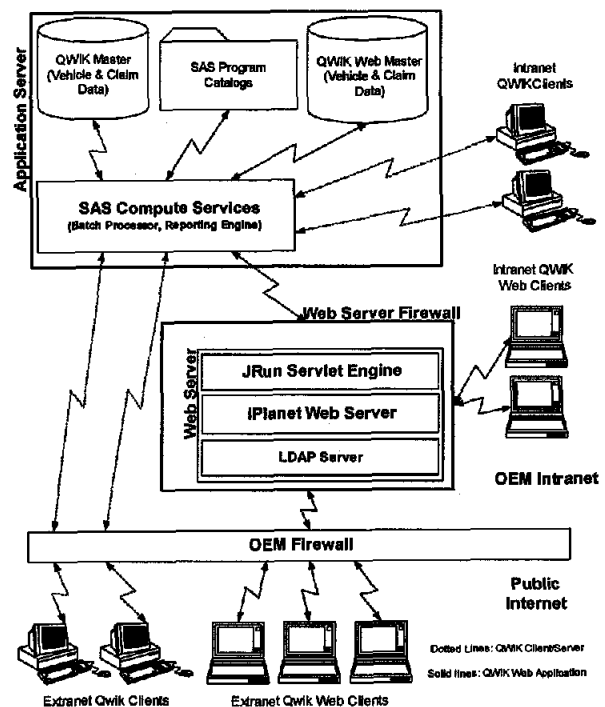
4.1 Web Server, LDAP, JRUN, Sun Java 2 Platform (J2EE), JDBC, RMI, and SAS Java classes. The application server requires Version 8.2 under Solaris of the following SAS software products: Base SAS®, SAS/CONNECT®, SAS/SHARE®, SAS/AF®, SAS/STAT®, SAS/QC®, and SAS/GRAPH®. AppDev Studio® version 2.02 and Sun Forte 3.0 served as development environments.

The full-client edition of QWIK has fifty SAS/AF Frames containing 450 form elements. There are about 50,000 lines of SAS code that split execution cycles between client and server. The new QWIK Web application serves up fifteen pages containing 225 form elements. There are about 26,000 lines of Java, JSP, HTML, and SAS code that execute on web and application servers.

ARCHITECTURE

QWIK (full client) and QWIK Web (thin client) share the same application server and data sources. The full client interface is supplied by SAS/AF Frame entries, while the thin client interface consists of a browser that displays HTML pages sent by the web server.

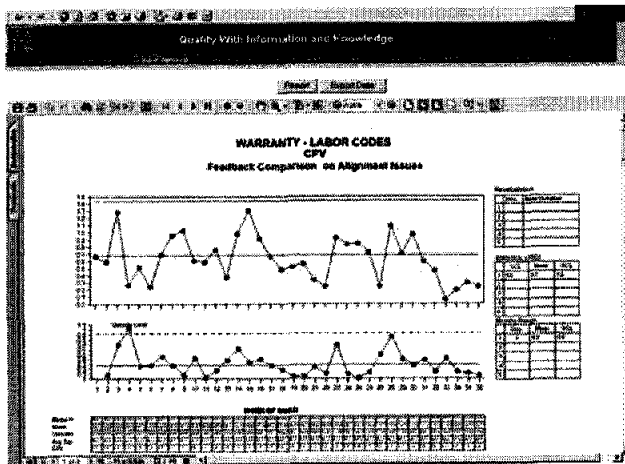
QWIK Production Environment



The web server houses QWIK Web classes, JavaServer Pages, form design elements, and style sheets. The application server houses SAS programs, reference tables, and production files.

SECURITY

The OEM Intranet does not operate like the typical Internet environment. Communication for authentication is unidirectional, and the messages sent are "Dead on Arrival". This causes authentication problems. In addition, when the application server



Report options include a top issue chart, control charts, reports on claim incidence and cost by category, and detail listings that drill down to specific vehicles and associated claims.

JAVA GLOSSARY

Java uses the *package.class.method(parameters)* statement style. This syntax is common in object oriented languages like SAS/AF, C++, C#, VB.NET, Eiffel, and Smalltalk. Any new object is defined as an instance of a class in the Java class hierarchy. Each new class inherits the attributes and behaviors of its parent, and can subsequently be modified to suit a particular purpose. Classes reside in packages that must be identified at the top of each Java program. Here are definitions for a few of the many web development constructs.

HTTP: HyperText Transfer Protocol v 1.1 is a request-response protocol consisting of a request method (e.g., GET, POST), a URI, header fields, and a body. HTTP Servlets written in Java manage state (sessions) on top of stateless HTTP, process data submitted by an HTML form, and provide dynamic content to the user through a browser.

HTML: HyperText Markup Language v 4.0 is the publishing language of the World Wide Web. HTML provides for publication of text, media, hyperlinks, scripting languages, style sheets, and printing facilities.

J2EE: The Sun Microsystems Java 2 Platform, Enterprise Edition, v 1.3.1 API specification supplies packages for writing Java applications.

Servlet: Java Servlets v 2.2 are modules developed by a programmer that run in a server application to answer client requests. Servlets make use of the Java packages "javax.servlet" and "javax.servlet.http".

JSP: JavaServer Page v 1.1 technology is an extension of servlet methods. The JSP combines static HTML templates with code for generating dynamic content.

Java Classes: In object oriented design, a *class* is the set of all programming objects that share common attributes and behaviors. *Methods* are program behaviors that act upon objects. A character variable is an *instance* of a class, while a trim operation on this variable's value would exemplify use of a method on the object.

JavaScript: JavaScript is an interpreted language that serves to extend HTML by handling events on the client side. Error trapping, messaging, and dialogs are often implemented in JavaScript.

RMI: Java Remote Method Invocation enables the programmer to create distributed applications in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts.

ROCF: The SAS Remote Object Class Factory is the entity which, given a connection and an interface to a model, creates the remote object and returns a handle to the remote object back to the viewer. For example, ROCF can serve as interface between a model SCL object on the application server and a Java object on the web server. Remote objects can be used, via the model interface, just like any other Java object. Client side Java objects include applets, servlets, Java beans, and JavaServer Pages.

JAVA CODE

QWIK Web application flow consists of :

- starting the application
- caching form element values
- providing for selection of values
- error handling
- processing and storing selections
- sending selections to the application server
- producing a report

"Model Year" is among the vehicle attributes required for a report to run. The following code fragments trace the life of "Model Year" as a list box selectable item.

QWIK Web application classes extend Java and SAS/webAF classes. These are stored in the "tti" package. All *.java files are compiled into *.class files, and are made available to one another by "import" statements in Java programs and "include" directives in JSP programs.

StartServlet.java initiates population of application controls.

```
// Identify resources
package tti;
import javax.servlet.http.HttpServlet;

public class StartServlet extends HttpServlet{
// QWIKCommonData caches application data
tti.QWIKCommonData tmpData=new
tti.QWIKCommonData();
}
```

List and drop-down boxes require values supplied by SAS data sets resident on the application server. *QWIKCommonVariables.java* creates string variables representing these data sets.

```
// Identify the Model Year data source
public interface QWIKCommonVariables{
public final String TABLE_MODEL_YEAR=
"Qwikdata.Web_years";
}
```

The table "ModelYears" is associated with the "Qwikdata" libref in the SAS autoexec on the application server. This table contains a code value for model year ("vinyr") in the first column and associated description in the second column ("modelyr"). The following routine iterates through the SAS data set in loading two Java collection objects: "staticModelYearCodes" and "staticModelYearDescriptions".

QWIKCommonData.class imports data from the application server

```
// Identify resources
import tti.*;
import java.io.*;
import com.sas.*;
// Establish application server connection
com.sas.rmi.Rocf
roc=new com.sas.rmi.Rocf();
com.sas.rmi.Connection
con=new com.sas.rmi.Connection();
con.setPort(tti.QWIKCommonVariables.port);
// Create a remote object on the app server
si=(SubmitInterface)roc.newInstance
(SubmitInterface.class.con);
dataset=(com.sas.sasserver.dataset.
DataSetInterface)roc.newInstance
(com.sas.sasserver.dataset.
```

```

        DataSetInterface.class, con);

// Cache Model Year codes (values) and
// descriptions
private com.sas.collection.StringCollection
staticModelYearValues;
private com.sas.collection.StringCollection
staticModelYearDescriptions;

try{
    dataSet.setDataSet(TABLE_MODEL_YEAR);
    dataSet.setDisplayedColumns(new String[]
    {"vinyl", "modelyr"});
    iTotalNumberOfRows=dataSet.getRowCount();
    codes=new String[iTotalNumberOfRows];
    values=new String[iTotalNumberOfRows];

    for (iCurrentRow=0;
        iCurrentRow<iTotalNumberOfRows;
        iCurrentRow++){
        values[iCurrentRow]=(String)
        dataSet.getCell
        (iCurrentRow+1,1).toString().trim();
        descriptions[iCurrentRow]=(String)
        dataSet.getCell
        (iCurrentRow+1,2).toString().trim();
    }

    staticModelYearValues=new
    com.sas.collection.StringCollection
    (values);
    staticModelYearDescriptions=new
    com.sas.collection.StringCollection
    (descriptions);
    values=null;
    descriptions=null;
    iTotalNumberOfRows=0;
    iCurrentRow=0;

} catch (com.sas.table.TableException e) {
    Application_Logger.fatal
    ("Error in caching Model Year.");
}

// Instantiate public GET methods
public com.sas.collection.StringCollection
getModelYearValues(){
    return new
    com.sas.collection.StringCollection
    (staticModelYearValues.toStringArray());
}

public com.sas.collection.StringCollection
getModelYearDescriptions(){
    return new
    com.sas.collection.StringCollection
    (staticModelYearDescriptions.
    toStringArray());
}

```

QWIKUserOptions.java is the central event processing module of the application. The following method (getModelYear) uses the SAS ListboxInterface method for storing selected Model Year codes and associated descriptions.

```

public com.sas.servlet.beans.ListboxInterface
getModelYear(){
    return createListbox
    ("selectModelYear", false,
    commonData.getModelYearValues(),

    commonData.getModelYearDescriptions(),
    userBean.collModelYear, 4);
}

```

Another QWIKUserOptions public method gets up to the minute user selections stored in the user bean.

```

// Public GET method
public com.sas.collection.StringCollection
getModelYearData() {
    return userBean.collModelYear;
}

```

analysis-step1.jsp contains HTML code for presenting users with the "Model Year" list box along with Java statements for handling selections.

```

<!--Identify resources-->
<% include file='../includes.jsp' %>

// Populate the list box
com.sas.servlet.beans.ListboxInterface
listboxModelYear=user_Options.getModelYear();

// Communicate selections
userOptions.setModelYear(request.
getParameterValues("selectModelYear"));

<!--Supply page content-->
<html>
    <br><hr noshade> Model Year: <br>
    <%
        listboxModelYear.setCustomAttributes
        (" multiple ");
        listboxModelYear.write(out);
    %>
</html>

```

QWIKUserBean.java stores user selections in Java collections.

```

// Import SAS string collection class
import com.sas.collection.StringCollection;

// Define a collection for selected values
public class QWIKUserBean() {
    public QWIKUserBean() {
        public com.sas.collection.StringCollection
        collModelYear=new
        com.sas.collection.StringCollection();
    }
}

```

analysis-step5.jsp sends selection parameters from the web server to the application server for processing against production files.

```

<!--Identify resources-->
<%include file="../includes.jsp" %>
<%@page import="com.sas.sasserver.*,
com.sas.sasserver.submit.SubmitInterface,
com.sas.rmi.Connection,
com.sas.rmi.Rocf,
com.sas.collection.StringCollection,
com.sas.collection.SortedCollection,
java.util.Properties,
java.sql.DriverManager,
java.sql.Driver"
contentType="text/html"
%>

// Establish connection to the app server
com.sas.sasserver.submit.SubmitInterface
si=null;
com.sas.sasserver.dataset.DataSetInterface
dataSet=null;
com.sas.rmi.Rocf roc=new com.sas.rmi.Rocf();
com.sas.rmi.Connection
con=new com.sas.rmi.Connection();
con.setServerArchitecture(con,"UNIX");
con.setHost(tti.QWIKCommonVariables.host);
con.setUsernamePrompt("login:");
con.setUsername(tti.QWIKCommonVariables.user);
con.setPasswordPrompt("Password:");
con.setPassword(tti.QWIKCommonVariables.pass);
con.setCommandPrompt
(tti.QWIKCommonVariables.prompt);
con.setCommandPromptTimeout
(tti.QWIKCommonVariables.timeout);
con.setCommand("/app/sas82/sas
-dmr -comamid tcp
-device grlink -noterminal -autoexec
/app/sas82/autoexe.sas -config
/app/sas82/sasv8.cfg -altlog
/qwikweb/saslog.log
-no\\$syntaxcheck");
con.setPort(tti.QWIKCommonVariables.port);
con.setFunnelHost
(tti.QWIKCommonVariables.host);

```

```
// Instantiate the SAS submit interface
si=(SubmitInterface)rocf.newInstance
  (SubmitInterface.class,con);
si.setProgramText(
  "options mprint mlogic symbolgen fullstimer;" +
  "libname qwikweb '/qwikweb' access=readonly;" +
  "filename pgms catalog 'qwikweb.webprograms';" +
  "%let ModelYear = %str(" +
  "    user_Options.getModelYearData().
    toString(',',false).replace(' ',' ') +
  ");" +
  "%include pgms(javabeen);" +
  "%javabeen;";
rocf.stop();
```

ROCF behaves like the SAS/CONNECT remote submit, and the submit interface looks like a submit block in SAS/AF SAS Component Language. In addition to using macro variables, the developer can use an SCL object or JDBC to communicate Java variable values to SAS on the application server side.

To complete the cycle, here is sample SAS code for processing Model Year values selected by the QWIK Web warranty analyst.

```
%macro javabeen;
data work.javabeen;
  length ModelYear $ 20;
ModelYear=symget('ModelYear');
run;
%mend javabeen;
```

In sum, the web server starts by caching items, the user selects from these options, a parameter file passes to the application server to subset and report on production files, and the user retrieves results once processing completes.

CHOICE OF MODEL

QWIK Web, Enterprise Guide, and QWIK form a continuum from thin through integrated object to full client technologies. Choice of technology depends on requirements. We suggest use of the following table in considering these options.

Requirements	Build with webAF	Use Enterprise Guide	Build with SAS/AF
Minimize development cost	No	Yes	No
Minimize maintenance cost	Yes	Yes	No
Minimize user training and support	Yes	No	Yes
Standardize reporting	Yes	No	Yes
Minimize response time	Yes	Yes	Yes

Minimize development cost: Commercial off the shelf software (COTS) is often preferred over custom development. If you don't have a budget for software development, then Enterprise Guide is a great tool for communicating with a Version 8 SAS Server. However, if you have very specific requirements that cannot be addressed by a reporting tool, then development could proceed using webAF for a web based solution, or SAS/AF as a client server solution.

Minimize maintenance cost: The only maintenance required of a COTS product is upgrading to newer releases. webAF represents somewhat lower maintenance cost because most changes occur only on the server side. SAS/AF solutions can be the most robust, but require distribution of client side modules with each maintenance release.

Minimize user training and support: Both web and client server solutions provide a structured environment for the user. Enterprise Guide is very user friendly, but requires a relatively sophisticated audience. Opportunity for error is often greater when there is greater flexibility in defining elements of the reporting process.

Standardize reporting: Since the automotive OEM is sharing risk and reward with the supplier community on the basis of QWIK Web reports, it is essential that both parties have an opportunity to replicate findings without resorting to code review. Structured applications provide uniformity in results, all other things being equal.

Minimize response time: All three models offer efficient means of responding, whether target data reside in SAS data sets or data base tables on the server. QWIK and QWIK Web feature a batch queue facility that sets appropriate expectations regarding time to completion for tasks that are more resource (CPU and I/O) intensive.

Casual users benefit most from a webAF based solution. The Enterprise Guide solution is good for the slightly more adventurous analyst. And a full client solution based on the SAS/AF model provides for both structured interaction with an application, and free form exploration on the client side with subsets drawn down from the server.

CONCLUSION

Providing web access to corporate data in a securely managed environment is the direction that all companies are exploring today. Web enabled applications are platform independent, providing access to many more users than the full client alternative. The user need only have a web browser to access a complex analytical production application that provides summary and decision support feedback based on user selections. It demonstrates that a SAS based application can be distributed over the web, without losing significant functionality. Further, it represents a cost-effective solution to the problem of sharing corporate data with suppliers while maintaining security.

REFERENCES

- Bailey, Chris and Moss, Karl (1999), "Distributing SAS/AF Models with Java Clients", *Proceedings of the Twenty-fourth Annual SAS Users Group International Conference* (<http://www2.sas.com/proceedings/sugi24/AppDevel/p08-24.pdf>)
- Barnes Nelson, Gregory S. (1999), "Extending the Life of Your AF Application Exploiting the Model-Viewer Paradigm", *Proceedings of the Twenty-fourth Annual SAS Users Group International Conference* (<http://www2.sas.com/proceedings/sugi24/Internet/p179-24.pdf>)
- Boone, Bryan and Herber, Pat (2001), "Delivering Information Everywhere using JSP and SAS", *Proceedings of the Twenty-sixth Annual SAS Users Group International Conference* (<http://www2.sas.com/proceedings/sugi26/p178-26.pdf>)
- Brinsfield, Eric (2002), "Avoiding Entanglements: Migrating Applications to the Web", *Proceedings of the Twenty-seventh Annual SAS Users Group International Conference* (<http://www2.sas.com/proceedings/sugi27/p043-27.pdf>)
- Chapal, Scott E. and Jones, J. W. (2001), "Using WebAF for Data Entry and Editing in an Integrated Conservation Monitoring Data Management System", *Proceedings of the Twenty-sixth Annual SAS Users Group International Conference* (<http://www2.sas.com/proceedings/sugi26/p024-26.pdf>)
- Chen, Qing and Sipe, Lori (2002), "Creating a Web-based Application Utilizing JSP and SAS Java Bean", *Proceedings of the Twenty-seventh Annual SAS Users Group International Conference* (<http://www2.sas.com/proceedings/sugi27/p161-27.pdf>)
- Davis, Danielle and Chapman, Don (1999), "Developing Data-Driven Task-Oriented SAS Powered Programs in Java", *Proceedings of the Twenty-fourth Annual SAS Users Group International Conference* (<http://www2.sas.com/proceedings/sugi24/AppDevel/p27-24.pdf>)
- Davis, Michael (1998), "SCL for the Rest of Us: Nonvisual Uses of Screen Control Language", *Proceedings of the Twenty-third Annual SAS Users Group International Conference*

<http://www2.sas.com/proceedings/sugi23/Advttutor/p33.pdf>

Fuchs, Anton and Torr, Mark (2002), "AppDev Studio – A Roadmap", White Paper, Cary, NC: SAS Institute, Inc.

Henderson, Don (2002), "SAS Solutions and Emerging Technologies: An Oxymoron or Intuitively Obvious? You Decide", *Proceedings of the Twenty-seventh Annual SAS Users Group International Conference*
(<http://www2.sas.com/proceedings/sugi27/p163-27.pdf>)

Henderson, Donald J. (1998), "SAS/IntrNet Software: A Roadmap", *Proceedings of the Twenty-third Annual SAS Users Group International Conference*
(<http://www2.sas.com/proceedings/sugi23/Internet/p154.pdf>)

Kelly, Chip (2000), "Patterns for Producing Web-enabled Applications", White Paper, Cary, NC: SAS Institute, Inc.

LaChapelle, Carl and Gagliano, Tammy L. (2000), "Migrating Your SAS/AF_FRAME Applications to the Web", *Proceedings of the Twenty-fifth Annual SAS Users Group International Conference*
(<http://www2.sas.com/proceedings/sugi25/25/tw/25p173.pdf>)

SAS Institute, Inc., *AppDev Studio Developer's Site, Component API*
(<http://www.sas.com/rnd/appdev/webAF/api.htm>)

SAS Institute, Inc. (2001), *Getting Started with AppDev Studio, Second Edition*, Cary, NC: SAS Institute Inc.

SAS Institute, Inc. (1999), *SAS Web Tools: Developing Java Based Applications Course Notes*, Cary, NC: SAS Institute Inc.

SAS Institute, Inc. (2001), *SAS Web Tools: Developing JavaServer Pages and Servlets Using webAF Software Course Notes*, Cary, NC: SAS Institute Inc.

SAS Institute, Inc. (1999), *SAS Web Tools: Understanding Java in webAF Applications Course Notes*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1999), *SAS OnlineDoc, Version 8*, Cary, NC: SAS Institute, Inc.
(<http://v8doc.sas.com/sashtml/>)

Sun Microsystems, Inc., *Java 2 Platform, Standard Edition, v1.3.1, API Specification*
(<http://java.sun.com/j2se/1.3/docs/api/>)

Walters, Barbara and Chapman, Don (1999), "Comparison of CGI and Java Technology Provided in SAS/IntrNet Software", *Proceedings of the Twenty-fourth Annual SAS Users Group International Conference*
(<http://www2.sas.com/proceedings/sugi24/Internet/p181-24.pdf>)

Walters, Barbara and Chapman, Don (1998), "Overview of Java Components and Applets in SAS/IntrNet Software", *Proceedings of the Twenty-third Annual SAS Users Group International Conference*
(<http://www2.sas.com/proceedings/sugi23/Internet/p156.pdf>)

Waxman, Mickey and Hoyle, Larry (1999), "SAS webAF for Java Application Development, a First Sip", *Proceedings of the Twenty-fourth Annual SAS Users Group International Conference*
(<http://www2.sas.com/proceedings/sugi24/Handson/p155-24.pdf>)

ACKNOWLEDGEMENTS

Bob Hornak and Jack Arnold, as TTI customers, have given QWIK Web form through their vision and diligence. We also would like to thank Trillium Teamologies management, Gregory Stanalajczko and Jim Brennan, for their continual support during all phases of the project and vicissitudes encountered during its development.

CONTACT INFORMATION

Sandy Walters, Project Manager

Trillium Teamologies, Inc.
219 S. Main St., Ste. 300
Royal Oak, MI 48067
(248) 584-2080
swalters@trilliumteam.com
url: www.trilliumteam.com

Development Team
Mike Senger, Stephanos Antoniadis, Bruce Reading,
Ed Barker, and Lyle Dodge
Trillium Teamologies, Inc.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks of their respective companies.