Paper 100-31

# Delivering Intelligence for Real-Time Decision Making

Eric J. Hunley, SAS Institute Inc., Cary, NC
Richard Foley, SAS Institute Inc., Cary, NC

## ABSTRACT

Businesses continue to demand the need for the integration of data across their entire organization. They also expect to obtain intelligence from information provided on-demand or delivered proactively or in real-time. With all the hype, buzzwords, and compliance fears in the market it can be confusing to understand terms such as "on-demand," "real-time," and "active data warehousing."

This paper helps you to understand these terms. It also discusses how SAS and SAS® Data Integration can be used to integrate with real-time data sources (message queues, Web services, and service-oriented architectures) to deliver the information needed to make decisions that are based on intelligence at the right time.

## INTRODUCTION

Being more profitable than your competitors, yet, at the same time being compliant with government and industry regulations, is the primary goal of every business. Successfully achieving this goal involves getting new and better products to the market faster, having a more efficient supply chain, managing the customer lifecycle at every touch-point, and having timely accurate reporting. Therefore, providing timely and accurate information becomes a challenge to all IT managers given today's market conditions:

- Data volumes are constantly increasing as enterprises integrate bleeding-edge technology such as RFID tags, increase data collection such as point-of-sales data, expand use of new technology such as e-business usage, and comply with new regulatory requirements.
- New business processes have collapsed the response time that is needed to process and retrieve information. For example, a customer comes to a Web site to purchase a product. This simple process requires a credit check, inventory check, and possible promotional and customer validation, not to mention, updates to inventory, logistics, and master customer information.
- Company acquisitions and mergers continue to put a strain on IT as they attempt to integrate and migrate data from two totally different systems or corporate cultures into a single source of information on customers, products, and other data points throughout the organization. This also includes synchronizing these different systems to ensure that information is consistent, accurate, and trustworthy between the differing environments.
- Corporate compliance and government regulations have forced companies to not only keep records for longer periods of time, but to establish business processes that optimally, require real-time monitoring, such as security, risk, and fraud detection.

Furthermore, challenges for IT Managers in providing timely and accurate information also come from within their own enterprises. IT Managers face enormous pressure to create and implement a strategy for automating and integrating business processes not only internally, but also externally with business partners and customers. However, automating a business process often requires integrating both data management and business intelligence procedures, which are normally done in batch (usually requiring complex calculations and historical information), with operational systems that work in real-time. Compounding the situation is the volumes of data gathered from ever-increasing touch points. IT managers are faced with managing these independent components of an enterprise-wide information system:

- **Topic depth** -- including varying levels of historical information as well as numerous attributes and analytics required. For example, marketing could require five years of Point-of-Sale data.
- **Integration breadth** -- including integrating internal and external business functions that previously were treated as completely independent functions, for example, inventory and marketing.
- **Timeliness of accurate information** -- quickly providing intelligence on information for efficient execution of business processes while constrained by the same resources and budget.

These constraints create a paradox when trying to add in-depth insight from more varied sources of information in a faster time frame. If more sales information is needed, then more data is required to better forecast sales events.

This is going to require more processing time, thus leaving a choice of changing delivery time or removing an integration touch point.

As many IT managers are being tasked to address the above mentioned independent components, it is not uncommon to hear them reference or pursue topics such as “active data warehousing” or “real-time” or “on-demand.” These topics sometimes mean very similar things, but are really defined in the eyes of the beholder. One author of this paper has heard “real-time” explained as anything that is faster than you can currently do it today or referred to as “right-time.” *Real-time* is the right amount of time to deliver relevant information defined by the consumer. For one consumer it can be a low-latency drip feeding of information as soon as it is available. For another consumer it might be to process information once it reaches a certain volume. Another consumer might define real-time or right-time as once a week. Again, real-time is defined by you, the consumer, and is based on the demands that your business function requires. So do not get caught up in all the hype or buzzwords. Determine what is required for you and leverage the technologies and capabilities that are available to deliver the relevant information in the time that you need it to make business decisions based on intelligence.

## SAS REAL-TIME ARCHITECTURE

Solving the paradox in a real-time enterprise requires an architecture that enables the rapid deployment of new business processes, while bringing the intelligence of decision making into the operational arena. The infrastructure needed to support information delivery in a real-time enterprise is the domain of middleware:

- **Message-oriented middleware** (see Appendix A “Message Queue Architecture”) provides asynchronous communication between applications, enabling applications to receive information, act on it, and, if a response is required, transmit a message to the designated destination—with no dependency on another application’s execution status.
- **Web services** (see Appendix B “Web Services Architecture”) are based on a set of industry standards and provide basic messaging and service-description functions via Internet protocols.

These middleware tools are the building blocks of a service-oriented architecture (SOA).

The complete realization of the real-time enterprise requires integrating middleware with operational and business intelligence environments. However, this integration goes beyond just sending and receiving information. In a real-time enterprise, decisions are made and expected at lighting speed and the accuracy and consistency of the information becomes even more critical as the information is being used to make critical business decisions without the option of timely checks and balances. SAS provides the ability to do the following:

- integrate with messaging middleware with open standard API’s
- rapidly create and deploy Web services using open standards
- apply data quality in both the operational and business intelligence environments to ensure accuracy of information at all touch points
- reduce the volumes of data required to process by only evaluating the records that have changed (change data capture)
- provide specialized storage designed for rapid retrieval and analysis of information

This section examines the same scenario in both a Message Queue Architecture and in a Web Services Architecture.

### SCENARIO OVERVIEW

The scenario is to read data sent to a message queue or streamed in through a Web service. The data can come from a call center, Web form, or any other application. The data is processed, cleansed, and integrated with other data, a data mining score is created, and the information is sent back as output.

### SAS WITH MESSAGE QUEUES

SAS can be triggered via a listener such as Platform Computing’s LSF software, or SAS can poll the message queue directly. Either way a connection is made between the message queue and SAS using the Application Messaging Interface (AMI). SAS Integration Technologies is the vehicle for SAS calling message queues through AMI. This approach enables the execution of the desired logic in both the SAS Workspace Server and the SAS Stored Process Server. Data can be read in or written to a queue, and for this scenario SAS will read in a file (.csv, XML, etc.) from the queue. Once read, the data can be parsed, any business rules or transformation can be applied, and then the

data can be scored. SAS will then reconnect to the message queue and send the information to the proper queue for retrieval by another application(s).

Figure 1 shows an example of a message queue flow diagram.

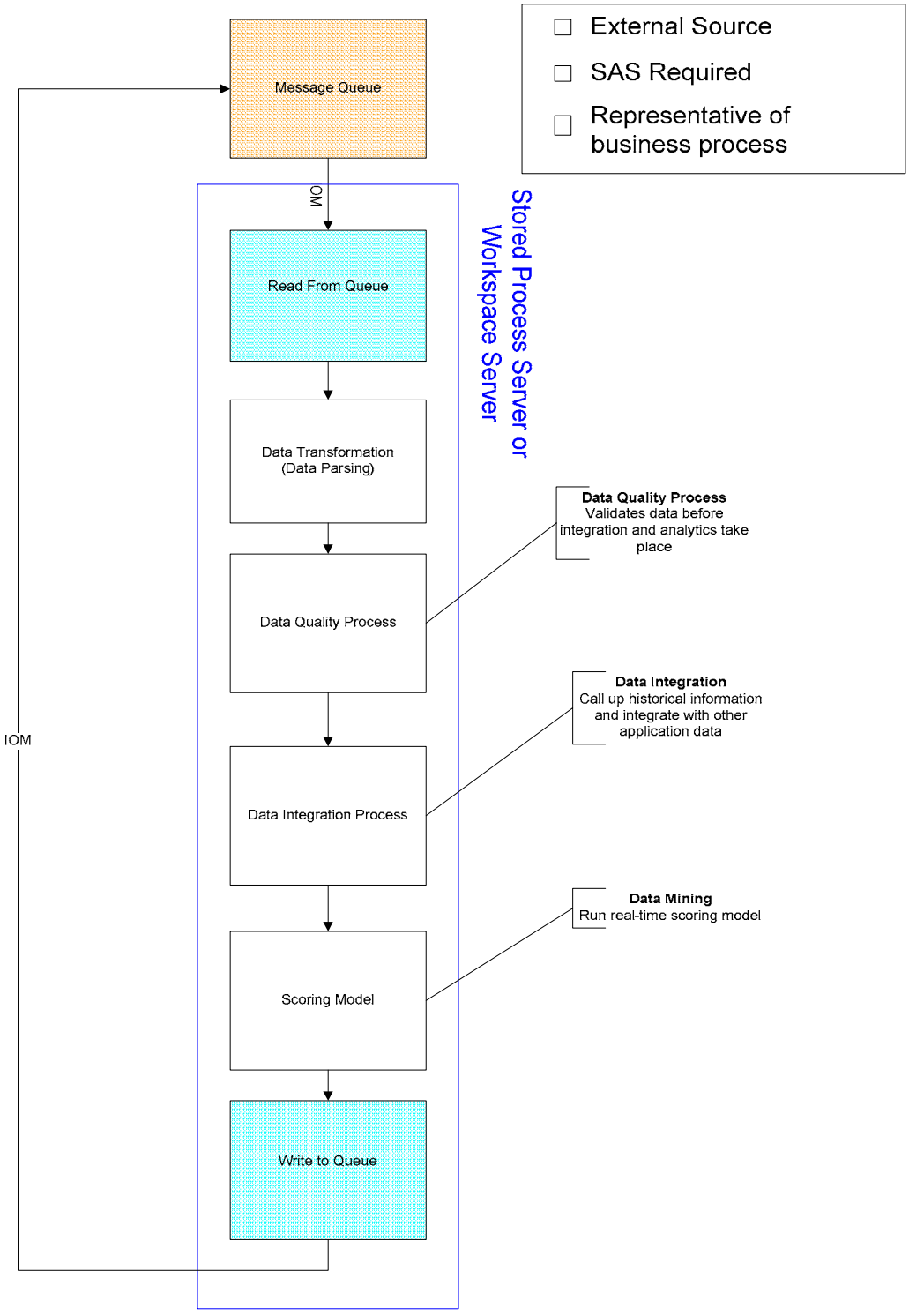



**Figure 1. Sample Message Queue Flow Diagram**

The following code is an example of some logic to read from a message queue (example taken from SAS Integration Technologies documentation).

```
/** Get a message from a queue **/

data _null_;
length hconn hobj cc reason 8;
length rc hod hgmo hmd hmap msglen 8;
length parms $ 200 options $ 200 action $ 3 msg $ 200;

hconn=0;
hobj=0;
hod=0;
hgmo=0;
hmd=0;
hmap=0;

/** Connect to queue **/
put '---------------- Connect to QMgr --------------';
qmgr="TEST";
call mqconn(qmgr, hconn, cc, reason);
if cc ^= 0 then do;
  if reason = 2002 then do;
    put 'Already connected to QMgr ' qmgr;
  end;
  else do;
    if reason = 2059 then
     put 'MQCONN: QMgr not available... needs to be started';
    else
     put 'MQCONN: failed with reason= ' reason;
    goto exit;
  end;
end;
else put 'MQCONN: successfully connected to QMgr ' qmgr;


put '---------- Generate object descriptor ---------';
action="GEN";
parms="OBJECTNAME";
objname="TEST";
call mqod(hod, action, rc, parms, objname);
if rc ^= 0 then do;
  put 'MQOD: failed with rc= ' rc;
  msg = sysmsg();
  put msg;
  goto exit;
end;
else put 'MQOD: successfully generated object descriptor';

/** Open to queue **/
put '--------- Open queue object for input ---------';
options="INPUT_SHARED";
call mqopen(hconn, hod, options, hobj, cc, reason);
if cc ^= 0 then do;
  put 'MQOPEN: failed with reason= ' reason;
  goto exit;
end;
else put 'MQOPEN: successfully opened queue for output';

put '--------- Generate get message options --------';
call mqgmo(hgmo, action, rc);
```

```
if rc ^= 0 then do;
  put 'MQGMO: failed with rc= ' rc;
  msg = sysmsg();
  put msg;
  goto exit;
end;
else put 'MQGMO: successfully generated get message options';


put '--------- Generate message descriptor ---------';
call mqmd(hmd, action, rc);
if rc ^= 0 then do;
  put 'MQMD: failed with rc= ' rc;
  msg = sysmsg();
  put msg;
  goto exit;
end;
else put 'MQMD: successfully generated message descriptor';


put '------------ Generate map descriptor ------------';
desc1="SHORT";
desc2="LONG";
desc3="DOUBLE";
desc4="CHAR,,50";
call mqmap(hmap, rc, desc1, desc2, desc3, desc4);
if rc ^= 0 then do;
  put 'MQMAP: failed with rc= ' rc;
  msg = sysmsg();
  put msg;
  goto exit;
end;
else put 'MQMAP: successfully generated map descriptor';

/** Get Message from queue **/
put '------------ Get message from queue ------------';
call mqget(hconn, hobj, hmd, hgmo, msglen, cc, reason);
if cc ^= 0 then do;
  if reason = 2033 then put 'No message available';
  else put 'MQGET: failed with reason= ' reason;
  goto exit;
end;
else do;
  put 'MQGET: successfully retrieved message from queue';
  put 'message length= ' msglen;

  /* inquire about message descriptor output parameters */
  action="INQ";
  parms="REPORT,MSGTYPE,FEEDBACK,MSGID,CORRELID,USERIDENTIFIER,
         PUTAPPLTYPE,PUTAPPLNAME,PUTDATE,PUTTIME";

  length report $ 30 msgtype $ 10 feedback 8 msgid $ 48 correlid $ 48
         userid $ 12 appltype 8  applname $ 28 date $ 8 time $8;

  call mqmd(hmd, action, rc, parms, report, msgtype, feedback, msgid,
            correlid, userid, appltype, applname, date, time);
  if rc ^= 0 then do;
    put 'MQMD: failed with rc ' rc;
    msg = sysmsg();
    put msg;
  end;
```

```
  else do;
    put 'Message descriptor output parameters are:';
    put 'REPORT= ' report;
    put 'MSGTYPE= ' msgtype;
    put 'FEEDBACK= ' feedback;
    put 'MSGID= ' msgid;
    put 'CORRELID= ' correlid;
    put 'USERIDENTIFIER= ' userid;
    put 'PUTAPPLTYPE= ' appltype;
    put 'PUTAPPLNAME= ' applname;
    put 'PUTDATE= ' date;
    put 'PUTTIME= ' time;
  end;
end;


if msglen > 0 then do;
  /* retrieve SAS variables from GET buffer */
  length parm1 parm2 parm3 8 parm4 $ 50;

  call mqgetparms(hmap, rc, parm1, parm2, parm3, parm4);
  put 'Display SAS variables:';
  put 'parm1= ' parm1;
  put 'parm2= ' parm2;
  put 'parm3= ' parm3;
  put 'parm4= ' parm4;
  if rc ^= 0 then do;
    put 'MQGETPARMS: failed with rc= ' rc;
    msg = sysmsg();
    put msg;
  end;
end;
else put 'No data associated with message';

/** Close queue **/
exit:
if hobj ^= 0 then do;
  put '------------------ Close queue ----------------';
  options="NONE";
  call mqclose(hconn, hobj, options, cc, reason);
  if cc ^= 0 then do;
    put 'MQCLOSE: failed with reason= ' reason;
  end;
  else put 'MQCLOSE: successfully closed queue';
end;


if hconn ^= 0 then do;
  put '------------- Disconnect from QMgr ------------';
  call mqdisc(hconn, cc, reason);
  if cc ^= 0 then do;
    put 'MQDISC: failed with reason= ' reason;
  end;
  else put 'MQDISC: successfully disconnected from QMgr';
end;


if hod ^= 0 then do;
  call mqfree(hod);
  put 'Object descriptor handle freed';
end;
```

```
if hgmo ^= 0 then do;
  call mqfree(hgmo);
  put 'Get message options handle freed';
end;
if hmd ^= 0 then do;
  call mqfree(hmd);
  put 'Message descriptor handle freed';
end;
if hmap ^= 0 then do;
  call mqfree(hmap);
  put 'Map descriptor handle freed';
end;


run;
```

**SAS WITH WEB SERVICES**

SAS uses SAS Web Services for .NET or SAS Web Services for Java to launch stored processes through a Web service. The SAS Web Services use the commands DISCOVER and EXECUTE from the XMLA open standard to return information. Calling DISCOVER returns all of the stored processes with the keyword "XMLA Web Service." Calling EXECUTE launches the stored process requested in the EXECUTE call. When setting up the stored process the following information needs to be taking into consideration.

- Because XML is being used it is recommended to create an XML map using the SAS XML Mapper. An XML map enables SAS XML LIBNAME to parse XML documents into various data sets. Another advantage is that the map stores information in metadata and can be used by the SAS Metadata Server without having to hand code a parsing routine. Removing hand coding decreases errors and in this case allows for easier maintenance of ever-changing XML schema's.
- For streaming XML out use the reserved fileref name _WEBOUT in the XML LIBNAME engine.
- To register a stored process to be used in a Web service the keyword "XMLA Web Service" must be used.

Figure 2 shows an example of a Web service flow diagram:

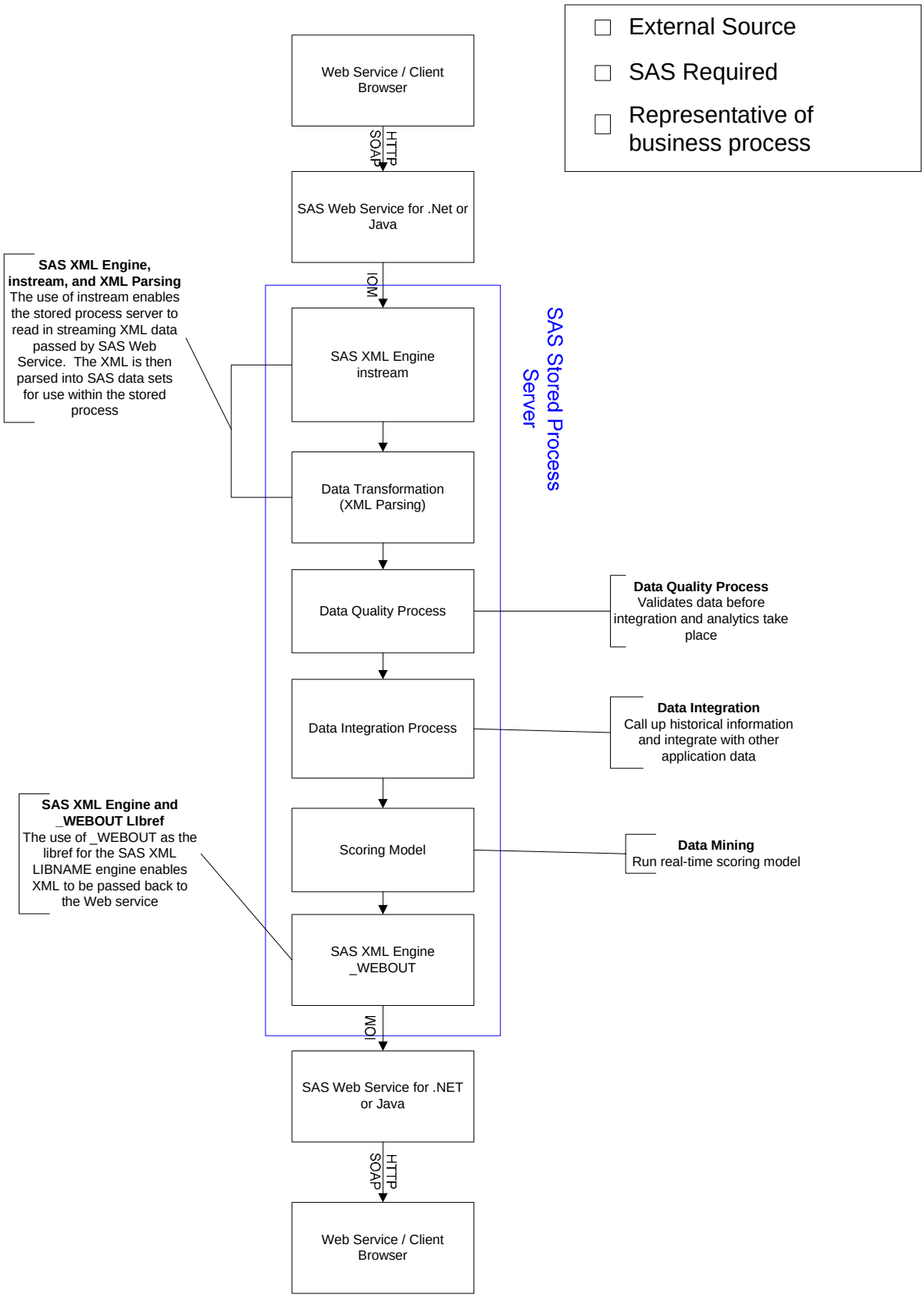



**Figure 2. Sample Web Service Flow Diagram**

The following code provides an example of some logic to read from a message queue (example taken from SAS Integration Technologies documentation).

```
/** Get a message from a queue **/

data _null_;
length hconn hobj cc reason 8;
length rc hod hgmo hmd hmap msglen 8;
length parms $ 200 options $ 200 action $ 3 msg $ 200;

hconn=0;
hobj=0;
hod=0;
hgmo=0;
hmd=0;
hmap=0;

/** Connect to queue **/
put '---------------- Connect to QMgr --------------';
qmgr="TEST";
call mqconn(qmgr, hconn, cc, reason);
if cc ^= 0 then do;
  if reason = 2002 then do;
    put 'Already connected to QMgr ' qmgr;
  end;
  else do;
    if reason = 2059 then
     put 'MQCONN: QMgr not available... needs to be started';
    else
     put 'MQCONN: failed with reason= ' reason;
    goto exit;
  end;
end;
else put 'MQCONN: successfully connected to QMgr ' qmgr;


put '---------- Generate object descriptor ---------';
action="GEN";
parms="OBJECTNAME";
objname="TEST";
call mqod(hod, action, rc, parms, objname);
if rc ^= 0 then do;
  put 'MQOD: failed with rc= ' rc;
  msg = sysmsg();
  put msg;
  goto exit;
end;
else put 'MQOD: successfully generated object descriptor';

/** Open to queue **/
put '--------- Open queue object for input ---------';
options="INPUT_SHARED";
call mqopen(hconn, hod, options, hobj, cc, reason);
if cc ^= 0 then do;
  put 'MQOPEN: failed with reason= ' reason;
  goto exit;
end;
else put 'MQOPEN: successfully opened queue for output';

put '--------- Generate get message options --------';
call mqgmo(hgmo, action, rc);
```

```
if rc ^= 0 then do;
  put 'MQGMO: failed with rc= ' rc;
  msg = sysmsg();
  put msg;
  goto exit;
end;
else put 'MQGMO: successfully generated get message options';


put '--------- Generate message descriptor ---------';
call mqmd(hmd, action, rc);
if rc ^= 0 then do;
  put 'MQMD: failed with rc= ' rc;
  msg = sysmsg();
  put msg;
  goto exit;
end;
else put 'MQMD: successfully generated message descriptor';


put '------------ Generate map descriptor ------------';
desc1="SHORT";
desc2="LONG";
desc3="DOUBLE";
desc4="CHAR,,50";
call mqmap(hmap, rc, desc1, desc2, desc3, desc4);
if rc ^= 0 then do;
  put 'MQMAP: failed with rc= ' rc;
  msg = sysmsg();
  put msg;
  goto exit;
end;
else put 'MQMAP: successfully generated map descriptor';

/** Get Message from queue **/
put '------------ Get message from queue ------------';
call mqget(hconn, hobj, hmd, hgmo, msglen, cc, reason);
if cc ^= 0 then do;
  if reason = 2033 then put 'No message available';
  else put 'MQGET: failed with reason= ' reason;
  goto exit;
end;
else do;
  put 'MQGET: successfully retrieved message from queue';
  put 'message length= ' msglen;

  /* inquire about message descriptor output parameters */
  action="INQ";
  parms="REPORT,MSGTYPE,FEEDBACK,MSGID,CORRELID,USERIDENTIFIER,
         PUTAPPLTYPE,PUTAPPLNAME,PUTDATE,PUTTIME";

  length report $ 30 msgtype $ 10 feedback 8 msgid $ 48 correlid $ 48
         userid $ 12 appltype 8  applname $ 28 date $ 8 time $8;

  call mqmd(hmd, action, rc, parms, report, msgtype, feedback, msgid,
            correlid, userid, appltype, applname, date, time);
  if rc ^= 0 then do;
    put 'MQMD: failed with rc ' rc;
    msg = sysmsg();
    put msg;
  end;
```

```
  else do;
    put 'Message descriptor output parameters are:';
    put 'REPORT= ' report;
    put 'MSGTYPE= ' msgtype;
    put 'FEEDBACK= ' feedback;
    put 'MSGID= ' msgid;
    put 'CORRELID= ' correlid;
    put 'USERIDENTIFIER= ' userid;
    put 'PUTAPPLTYPE= ' appltype;
    put 'PUTAPPLNAME= ' applname;
    put 'PUTDATE= ' date;
    put 'PUTTIME= ' time;
  end;
end;


if msglen > 0 then do;
  /* retrieve SAS variables from GET buffer */
  length parm1 parm2 parm3 8 parm4 $ 50;

  call mqgetparms(hmap, rc, parm1, parm2, parm3, parm4);
  put 'Display SAS variables:';
  put 'parm1= ' parm1;
  put 'parm2= ' parm2;
  put 'parm3= ' parm3;
  put 'parm4= ' parm4;
  if rc ^= 0 then do;
    put 'MQGETPARMS: failed with rc= ' rc;
    msg = sysmsg();
    put msg;
  end;
end;
else put 'No data associated with message';

/** Close queue **/
exit:
if hobj ^= 0 then do;
  put '------------------ Close queue ----------------';
  options="NONE";
  call mqclose(hconn, hobj, options, cc, reason);
  if cc ^= 0 then do;
    put 'MQCLOSE: failed with reason= ' reason;
  end;
  else put 'MQCLOSE: successfully closed queue';
end;


if hconn ^= 0 then do;
  put '------------- Disconnect from QMgr ------------';
  call mqdisc(hconn, cc, reason);
  if cc ^= 0 then do;
    put 'MQDISC: failed with reason= ' reason;
  end;
  else put 'MQDISC: successfully disconnected from QMgr';
end;


if hod ^= 0 then do;
  call mqfree(hod);
  put 'Object descriptor handle freed';
end;
```

```
if hgmo ^= 0 then do;
  call mqfree(hgmo);
  put 'Get message options handle freed';
end;
if hmd ^= 0 then do;
  call mqfree(hmd);
  put 'Message descriptor handle freed';
end;
if hmap ^= 0 then do;
  call mqfree(hmap);
  put 'Map descriptor handle freed';
end;


run;
```

## SAS IN A REAL-TIME ENVIRONMENT

SAS is flexible enough to work with either message queues or Web services, but each is handled a little differently due to the nature of the architecture (see Appendix A “Message Queue Architecture” and Appendix B “Web Services Architecture”). Web services requires communication via HTTP and message queues use a different protocol all together. This section describes how individual SAS offerings enable you to quickly create business processes that utilize and take advantage of message queues or Web services or both.

SAS within a service-oriented architecture is able to do the following:

- increase organizational effectiveness by automating business processes to provide information when it’s needed
- provide the ability to expose services to parties outside of your organization. Customers, suppliers, and partners can easily integrate these services into their own systems.
- speed the development and deployment of applications, reducing the time to market and overall business costs
- increase return on SAS investments with systems that are based on standards agreed to by all major software vendors.

### DATA INTEGRATION AND MANAGEMENT

The closer to the operational process a data transformation takes place the better for the real-time enterprise. The placement of certain data management and data integration processes within a service-oriented architecture can have a large cascading effect upstream of the business process, yielding benefits such as the following:

- increased ability to support more data depth as many of the data integration processes have been moved closer to the operational system
- data integration in real-time through data transformations and the loading of one operational system to another without the use of batch processing
- better data management as standards can be used that enable more consistency throughout the information value chain
- increased ability to quickly enrich information in a business process from multiple sources, thus improving the breadth of information the process can draw upon to make intelligent decisions

Therefore, in a real-time environment, integrating various data integration processes into a business processes is required. These processes can be as simple as transforming a date from a mmddyy (Month, Day, Year) to a ddmmyyyy (Day, Month, Year) format or a currency conversion for integrating operational systems, or it can be as complex as parsing, cleansing, and analyzing a corporate financial statement as part of a credit risk solution. The easier it is to take the various processes and convert them into real-time business processes the more flexible the system becomes, thus satisfying one of the requirements to resolve the paradox of the real-time enterprise. SAS Data Integration Studio provides the ability to create jobs that can be deployed as SAS Stored Processes, which runs SAS code as a self-contained service with explicitly defined parameters. Through the use of the SAS Data Integration Server these stored processes can work with message queues or be called by a Web service, thus

enabling the total integration of SAS into a service-oriented architecture. (Message queues can also work with the SAS Workspace Server.)

In SAS Data Integration Studio one way to implement a Web service is to create a job that reads XML via the XML LIBNAME engine (using an XML map produced by the XML Mapper utility) and parses that information into the desired format (Figures 3 and 4).

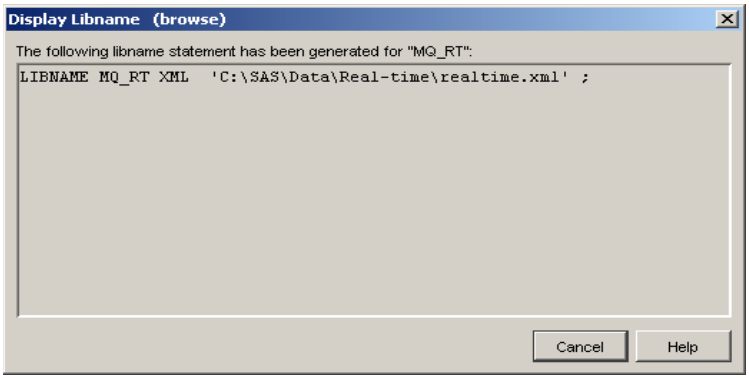



**Figure 3. LIBNAME for Defined XML File**

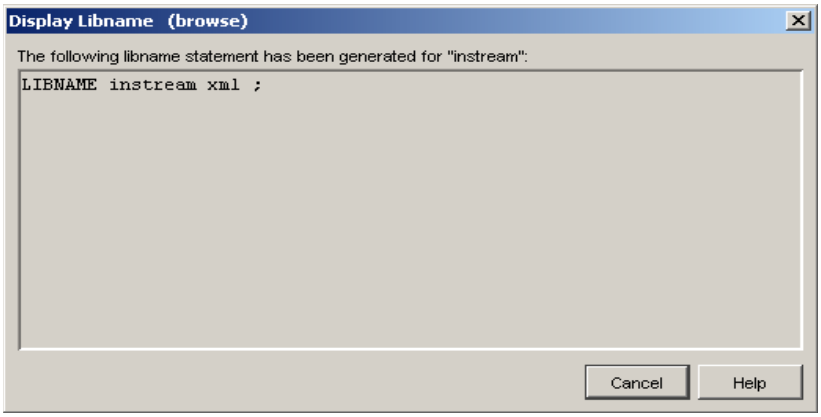



**Figure 4. LIBNAME for an XML Stream**

The next step is to create the flow of the desired processes (Figure 5), whether you are applying a data cleansing business rule, scoring a record based on a pre-defined score logic, or forecasting a customer's likelihood to purchase a product based on current information.

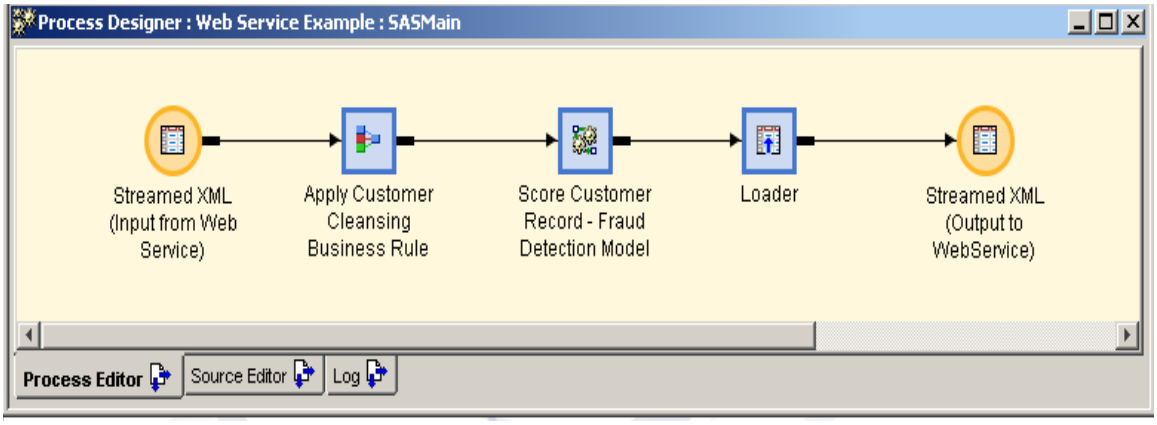



**Figure 5. Real-Time Process for Cleaning and Scoring a Customer Record**

Finally, an XML response is usually the desired outcome and again this would require the use of the generic LIBNAME engine in SAS Data Integration Studio and using the XML engine to create XML output. Note that the "_webout" is a required libref for streaming XML output (Figure 6).

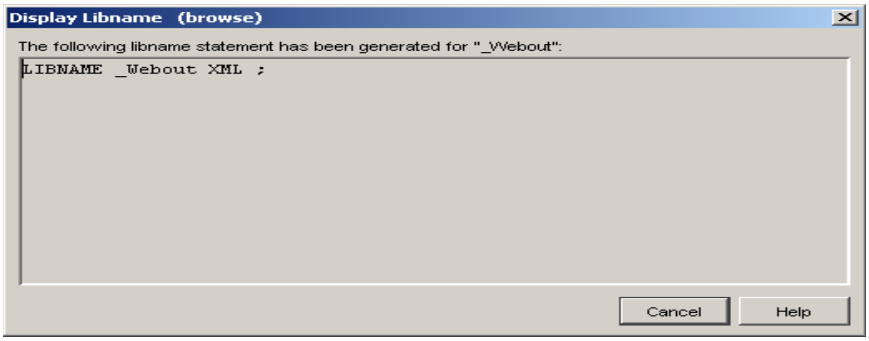



**Figure 6. LIBNAME for Streamed XML Output**

The next step in the process is to deploy the job as a stored process (Figure 7).

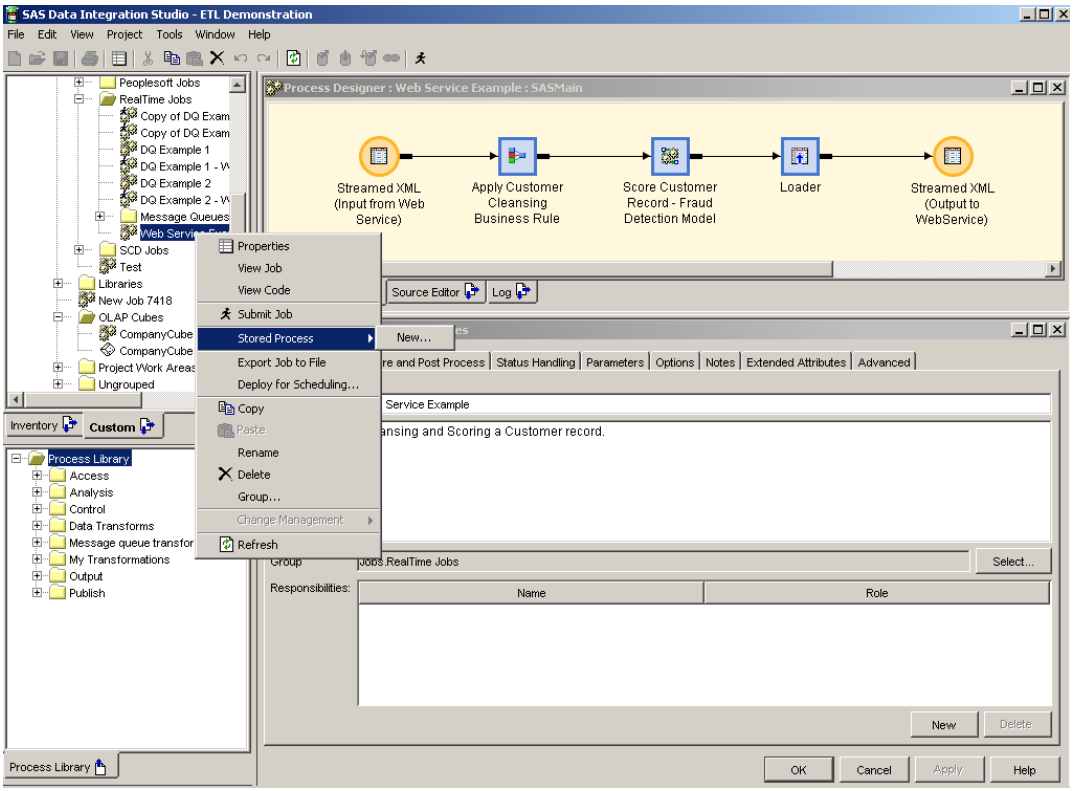



**Figure 7. Deploying a Stored Process in SAS Data Integration Studio**

Register the stored process as a Web service by using the keywords of "XMLA Web Service" (Figure 8).

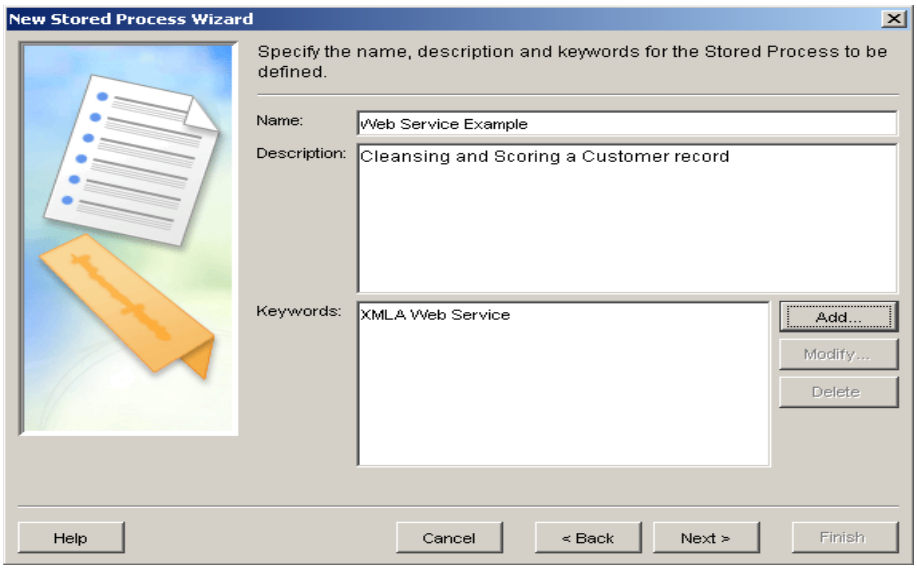



**Figure 8. Registering the Stored Process as a Web Service**

Define the Input and Output settings to "Streaming" to indicate that the stored process is expecting streamed input and will produce streamed output (Figure 9).

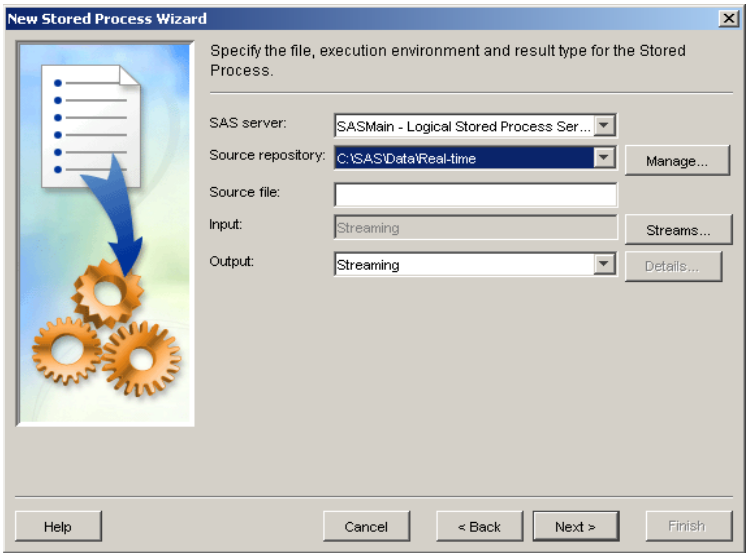



**Figure 9. Defining Input and Output Source**

The above process illustrates the simple steps of taking either a simple or complex data integration process or job into a service-oriented architecture via a Web service deployment.

**OPERATIONAL DATA QUALITY**

In a real-time environment information quality is often sacrificed for speed. This is usually at the expense of the corporation, thus nullifying any benefits received from implementing real-time processes. A successful real-time process, like any other business process, requires quality information. As stated above, without accuracy and consistency of all data, the trustworthiness of the information produced from it introduces risk throughout the organization. The earlier in the process the data can be fixed the faster the data can be processed and the more reliable the information throughout the process will be.

SAS and its subsidiary, DataFlux, have technology to implement data cleansing/quality rules in the operational environment, at the point of origin for inconsistent and inaccurate data, as well as in any downstream processes for managing data, business intelligence, and other business applications. The same jobs discussed above in the data integration process flow can be re-used as a service providing integration into a wide variety of software architectures including Internet or Intranet applications, client/server data entry and reporting applications, and other complex environments. These services can be called from within operational systems (Figure 10), as close as possible to the point of entry, ensuring that accurate information is used in making real-time decisions.

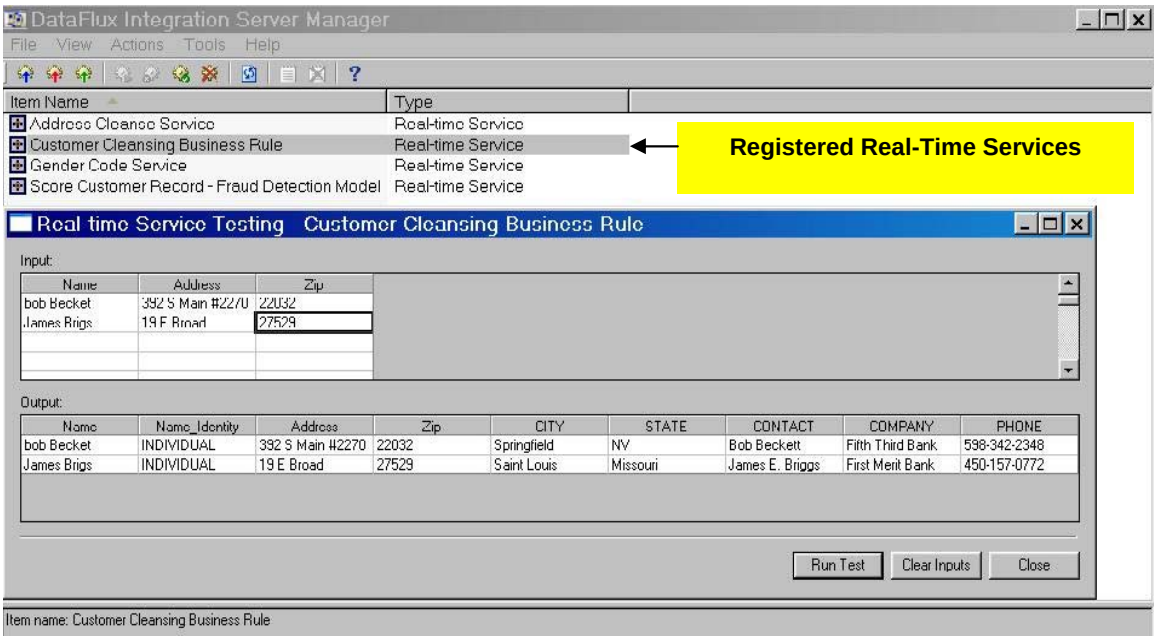



**Figure 10. Test of Real-Time Service for Customer Cleansing**

**ANALYTICS**

The real-time architecture requires bringing the intelligence of a critical decision support system closer to the operational systems to facilitate automation of a business process and add efficiencies to business processes such as the following:

- customer credit scoring
- inventory optimization
- real-time risk management
- supply chain optimization

This requires the delivery of analytical information to a live channel in a timely manner, usually sub-second for single records. As an example, message stream-based scoring is a very easy approach to implement and is also very scalable. SAS analytical technologies express scoring functions as Base SAS DATA step code. SAS Data Integration Studio and SAS Integration Technologies provides interfaces to standard message queue (MQ) middleware that are frequently the communication hubs of on-demand recommendation systems.

A real-time enterprise must be flexible and adaptive. So, it is important to distinguish the two primary tasks of any analytical modeling endeavor, model development (training), and model deployment (scoring). Model development typically requires assembling a historical data set(s) and applying several statistical techniques to estimate some quantity, for instance the probability of potential fraudulent activity. Numerous candidate models are often tried. The objective is to train a set of functions to produce the best model. Although very simple analytical models can be developed in real-time, SAS advocates developing more accurate and useful models in a batch environment. Offline model development enables the modeler to apply more sophisticated algorithms, such as neural networks, robust regressions, mixed models, and tree classifiers. It also allows for considerably more candidate input variables to be analyzed during model training. The task of applying a model to new data to produce a decision (recommendation) is known as model deployment, which can be done in batch or online or in real time.

Figure 11 provides an example that illustrates data being fed into a process that reads a customer record(s), cleanses the record, determines if it is an erroneous record or not, and then scores the record and writes it out to the appropriate message queue for additional processing.

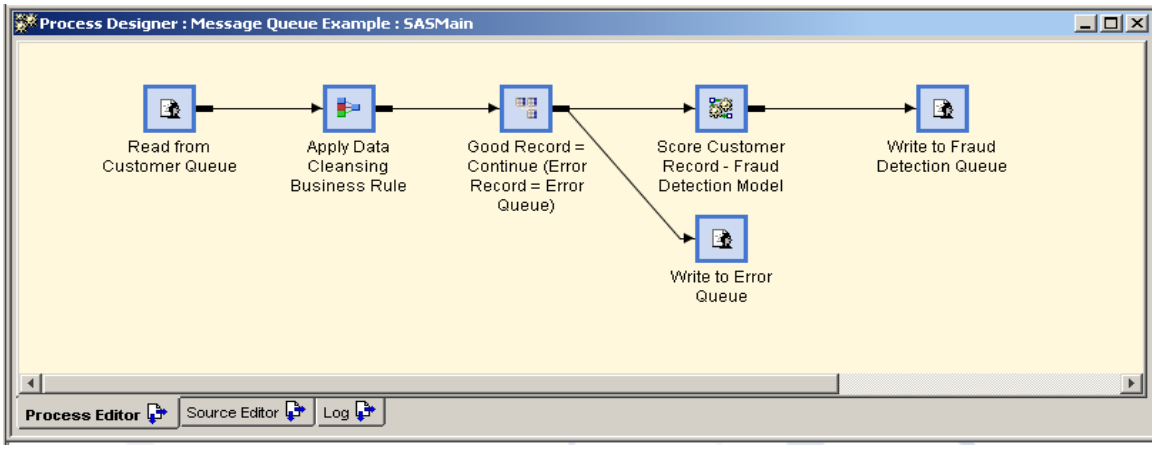



**Figure 11. Sample Message Queue Job Flow**

Similarly, this procedure can be set up as a Web service as described in the "Data Integration" section.

**REPORTING**

SAS Business Intelligence uses stored processes to not only execute processes and functions, but also to manage the procedures and function, by naming, storing, and securing them. This gives end users the flexibility to run the process in real-time mode--getting the most up-to-date information when they need it. For example, it is possible to create a simple or complex press (as defined above in the "Data Integration and Management" section) using SAS Data Integration Studio to define a stored process that is then executed in a SAS Business Intelligence environment. This provides the right information to the right user at the right time, a key to any BI infrastructure. Furthermore, SAS Business Intelligence provides the flexibility to have reports run on triggers such as an event to a message queue, run on demand, or run at scheduled time-based intervals.

SAS real-time reporting environment gives business users the necessary information to take proper action as fast as possible. Taking the proper action requires a reporting environment designed to notify you of potential issues, and provide you with the ability to dive deeper into what is causing the issues. The SAS Information Delivery Portal with built-in dashboards and alerting capability can immediately notify a user of changes in trends to key performance indicators or of passing user-defined thresholds. By having various portlets surface information from real-time stored processes, information maps, and SAS Web Report Studio reports you can quickly investigate what the issues are and take the best action based on the most current and accurate data.

## CONCLUSION

Doing business in the future will require technologies that enable an enterprise to adjust in many different ways. There's an increasing need for a more flexible and open architecture to successfully automate and integrate business processes to improve organizational effectiveness. A service-oriented architecture, such as Web services, can help meet that demand.

The power of SAS enables integration across your enterprise to provide real-time decision making. Based upon industry-standard technologies for messaging architecture and Web services, you can choose the technology that fits best within your organization to do the following:

- increase organizational effectiveness by automating business processes to provide information when it's needed
- provide the ability to expose services to parties outside of your organization. Customers, suppliers, and partners can easily integrate these services into their own systems.
- speed the development and deployment of SAS applications, reducing time to market and overall business costs

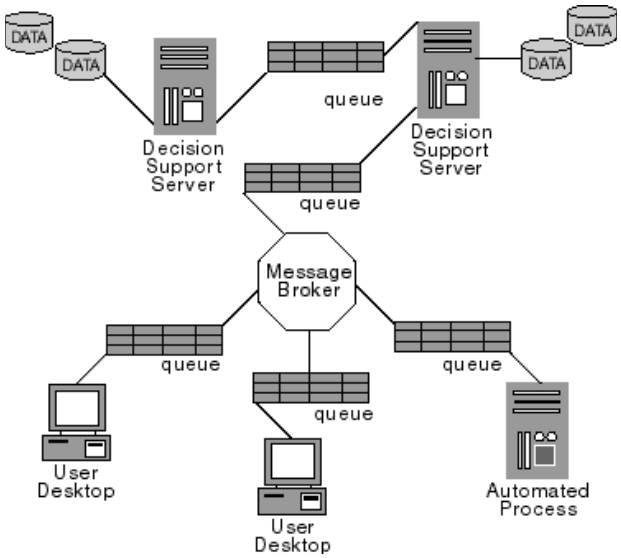
- increase long-term return on SAS investments with systems that are based on standards agreed to by all major software vendors.

The key to realizing the benefits of real-time decision making while staying above the hype cycle noise is to remember that real-time is defined by the business process, not an arbitrary time slice of data. This enables you to focus on the process and the solution to the process, which is why working with an architecture that enables you to chose what best fits your infrastructure without compromising on performance or quality of information is key to making real-time decisions and staying ahead of the competition in an ever-changing environment.

## APPENDIX A. MESSAGING ARCHITECTURE

In the world of corporate mergers and acquisitions, geographically dispersed offices, legacy applications, and ever-changing operational requirements, the scale of an integration project can become large and wide-ranging. Factors from simple time-zone differences to problematic communication links to changing system interdependencies can present seemingly insurmountable difficulties. The answer is an adaptive architecture that loosely couples various application systems to form integrated solutions. This is the domain of application-messaging middleware.

One feature of application-messaging middleware is asynchronous communication. A sending application can transmit a message without requiring the receiving application to be connected to the network, or even be active at the time of transmission. When the receiving application does go online, it can receive the pending message, act on it and, if a response is required, transmit a message to the designated destination—with no dependency on another application's execution status.




Adapters of three of the leading commercial messaging platforms (IBM Websphere MQ, Microsoft MSMQ, and TIBCO TIB/Rendezvous) are provided with SAS Integration Technologies. Access to these adapters is provided through application programming interfaces (APIs) that can be called from SAS. Applications developers who are already experienced with any of these messaging platforms can readily apply their existing knowledge because the APIs are modeled after the vendors' functional interfaces. Additionally, an operating environment-independent API is supported that enables portable applications to be developed over any of these messaging platforms.

With these messaging capabilities, SAS applications can be effectively integrated within an enterprise solution. For example, you might have a SAS program that produces inventory-forecasting information. Generally, the application might generate a report showing that a particular stock item needs to be reordered. An analyst would look at the forecast report, go to an order entry system, and place a reorder.

Processes can be automated using application messaging. The SAS analytical application runs in the background. When a certain threshold of stock depletion is reached, an event message is posted to the message queue giving information, such as the product ID number, a forecast of when the stock will be depleted, and what the standard reorder quantity is. In operational environments, a business system receives the order requests from the messaging queue, so instead of generating a batch report, a SAS analytical application automatically triggers the operational system to place a reorder.

The ability to leverage the power of SAS through industry-standard application-messaging middleware—from triggering back-office events out of data warehousing processes to exploiting analytical services from line-of-business applications—enables the creation of powerful enterprise solutions. The benefits of application-messaging middleware are as follows:

- Diverse autonomous systems can be connected to SAS, thus creating a full and accurate view of your information world no matter how latent they may be.
- Processes can be automated that were previously split between the operational and business intelligence worlds.
- Improved application communication is achieved throughout the entire supply chain.

For more information on the application-messaging support that is provided by SAS Integration Technologies, as well as sample code ,see the SAS Integration Technologies Release 8.2 Library at www.sas.com/rnd/itech/doc/messageq/index.html.

## APPENDIX B. WEB SERVICES

Web services are based on a set of industry standards and provide basic messaging and service-description functions via Internet protocols. Gartner DataQuest says, "Web services will change the face of integration solutions by taking them from being complex and expensive projects to relatively cheap and easy ones. This will have the effect of changing the market perception of integration projects and driving down costs" (Gartner 2002).

Web services are perfectly suited for delivering business intelligence and analytical services across the organization, as well as to users outside of the enterprise such as customers, suppliers or partners. Today, reaching users outside an enterprise is achieved mostly via extranets, which have proven effective in a number of business scenarios. Web services offer the potential to take Web-based business intelligence to the next evolutionary level by providing extranets with secure, standardized interfaces, technological elegance, easy integration, and economical repeatability, thus speeding the development and deployment of applications, reducing time to market, and decreasing overall business costs.

There are two implementations of SAS BI Web Services: one written in Java that requires a servlet container, and another written in C# using the .NET framework.

The SAS BI Web Services interface is based on the XML for Analysis (XMLA) Version 1.1 specification. XMLA is a standard specification developed by several companies for use as a Web service interface to access online analytical processing (OLAP) and data-mining functions. A client application can use this interface to invoke and get metadata about SAS Stored Processes.

The following figure shows how Web services work. A client, such as a Web application or desktop application, starts out by obtaining the Web Service Description Language (WSDL) from the Web service. The WSDL describes the methods available, endpoint (where to call the Web service), and the format of the XML required to call the Web service.

Web service clients and servers transport XML data using a data envelope called a SOAP envelope. Any client that can send and receive SOAP messages can access Web services. SAS supports SOAP bindings over HTTP. The client sends XML requests and parameters in a SOAP envelope to the Web service, telling the Web service to either Discover or Execute stored processes.


Discover() and Execute() are the two methods that are specified for XMLA. The Discover method consists of middle-tier code that calls the SAS Metadata Server to get the requested metadata. The Execute method consists of middle-tier code that calls the SAS Stored Process Server to invoke stored processes.

During execution, the requested stored process is executed by a SAS Stored Process Server. Usually any number of simple string parameters are passed to the stored process, and a stream of XML data is returned. The input

parameters to the stored process can be nothing or a combination of simple string parameters and any number of XML streams. The output from a stored process called by a SAS BI Web Service is always an XML stream.

## REFERENCES

Gartner. "*Gartner Dataquest Says the Application Integration, Middleware and Portal Markets Will See Steady Growth Through 2006.*" June 25, 2002. Available www.gartner.com/5_about/press_releases/2002_06/pr20020625a.jsp.

## ACKNOWLEDGMENTS

Thanks to all the contributors of information and reviews of this paper. Special thanks goes to Wayne Thompson, I-Kong Fu, Rick Styll and Lisa Flint and all the other resources that provided samples and documentation on this topic.

## CONTACTS

Your comments and questions are valued and encouraged. Contact the authors:

Eric J. Hunley
SAS Institute Inc.
100 SAS Campus Drive
Cary, NC 27513
Work Phone: 919-677-8000
Email: Eric.Hunley@sas.com

Richard Foley
SAS Institute Inc.
100 SAS Campus Drive
Cary, NC 27513
Work Phone: 919-677-8000
Email: Richard.Foley@sas.com