Paper 004-2011

Building Enterprise Applications Using SAS® Real-Time Services

Falko Schulz, SAS Institute Inc., Brisbane, Australia

ABSTRACT

SAS® Web services can help you meet the challenges of integrating the processes in your service-oriented architecture (SOA). Real-time services allow you to leverage the power of SAS across the enterprise and beyond. This paper demonstrates how to create SAS Web services and how to call these Web services from a third-party application. Increase your return on investment by integrating more processes in your SOA with SAS Web services.

Note: A zip file with code examples from this paper is available at http://www.sascommunity.org/wiki/Building_Enterprise_Applications_using_SAS_real-time_services.

INTRODUCTION

SAS 9.2 has excellent capabilities to create Web services that surface SAS advanced analytics. Using Web services standards ensures that these services are available to all applications in your SOA.

SAS BI Web services expose SAS stored processes as Web services, making it easy for other applications to call SAS code. A Web service is described by a Web Service Description Language (WSDL). WSDL is an XML file that describes the set of operations that a service contains as well as the inputs and outputs of each operation.

EXAMPLE SCENARIO

For the purpose of this document the following scenario is used:

There is the requirement to build a new call center interface to help call center operatives make the best possible tariff recommendations to customers. The interface should be able to retrieve the customer's details by providing the customer's phone number or ID. The operator should also be able to make changes to the customer's details. Finally, the interface needs to retrieve the customers churn score and make promotion offers in real time.

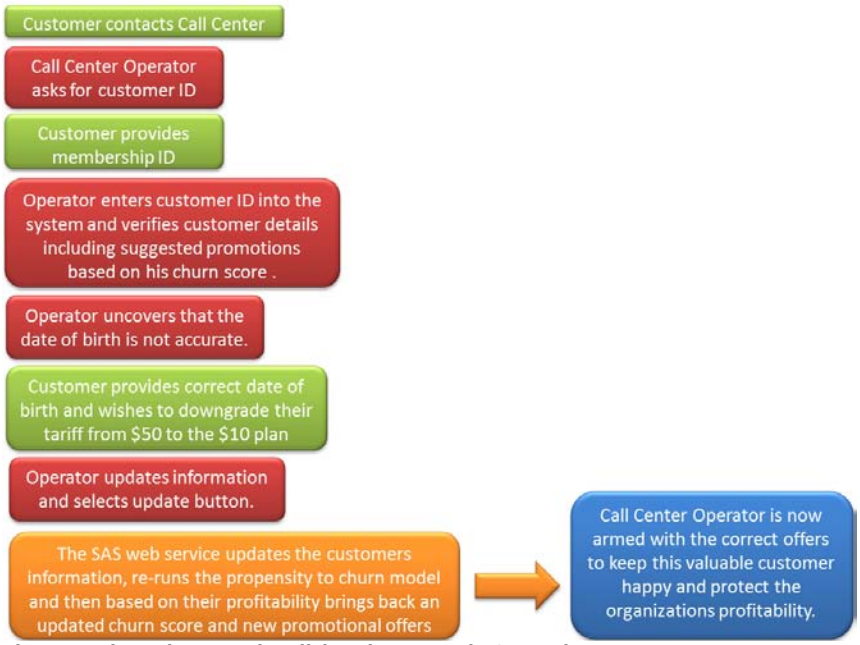


Figure 1. Flow Diagram Visualizing the Example Scenario

The following figure shows the proposed interface. Technology used to develop the user interface is Adobe Flex.

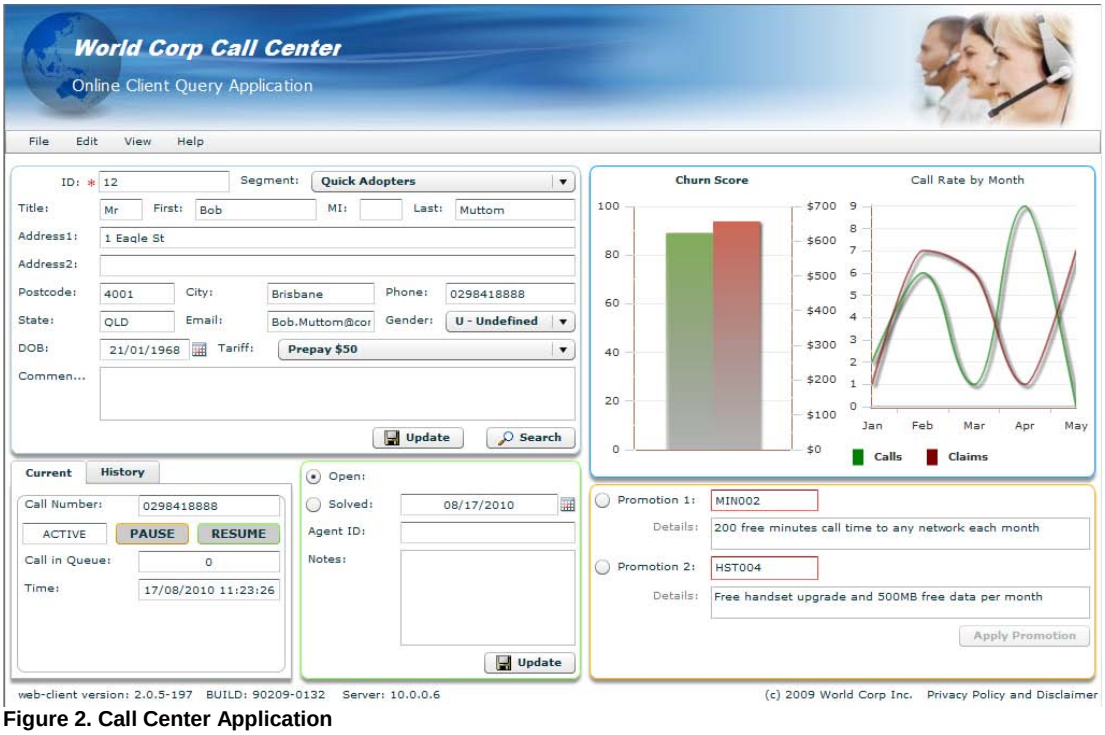


Figure 2. Call Center Application

CREATING THE SAS WEB SERVICE

SAS BI Web services are a part of SAS® Integration Technologies, which runs on the middle (Web) tier. Given the call center application design, there are two Web services that need to be built.

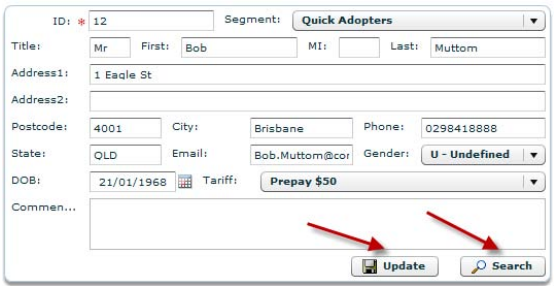


Figure 3. Customer Details

1. **Customer Detail Record Service** – a Web service returning the customer details by providing the customer ID or phone number
2. **Customer Update Service** – a Web service updating the customer detail record and returning the recalculated churn score

Each Web service will be executed by clicking either the **Update** or the **Search** button.

CUSTOMER DETAIL RECORD SERVICE

This Web service returns the customer details by a given customer ID or phone number. A service is a SAS program executed by the SAS® Stored Process Server. Parameters passed to this service are available as SAS macro variables and results of the SAS program are streamed back as XML as part of the Simple Object Access Protocol (SOAP).

This paper uses the following sample table as data source (you can download the full table at http://www.sascommunity.org/wiki/Building_Enterprise_Applications_using_SAS_real-time_services).

	custID	custTitle	custSegment	custFirstName	custLastName	custAddress1	custCity	custState	custPostcode	custPhone	custEmail	custGender	custDoB	custTariff	custProb	custLastMonthVal
1	1		E	Kenneth	Foreman	P.O. Box 562, 7657 Orci St.	Nevada City	NJ	54400	(745) 100-2227	scelerisque.mollis	F	01/22/1962	18	0	572
2	2	Dr.	B	Leslie	Crawford	496-5397 A, Av.	Waycross	Montana	82822	(426) 530-2889	a.facilisis@aliqua	F	08/24/1966	17	0	632
3	3	Dr.	C	Igor	Lott	493-6218 Facilisis Rd.	Salem	OK	8779	(155) 192-7676	non.ante@Nuncm	F	06/06/1979	8	1	691
4	4		C	Raphael	Gillespie	P.O. Box 256, 2192 Ante Rd.	New Britain	Colorado	55537	(808) 888-9647	per.inceptos@quis	M	06/18/1984	4	1	668
5	5	Dr.	E	Amy	Phillips	524-7754 Tellus, Avenue	Palo Alto	WA	43395	(205) 104-4473	sed.hendrerit.a@n	M	02/11/1961	12	0	685
6	6	Mr.	C	Jamalia	Chase	Ap #903-3093 Ullamcorper Avenue	Kenosha	MO	43877	(668) 487-4405	convallis.ante@au	M	12/06/1970	4	0	698
7	7		E	Ryan	Hatfield	665 Gravida Street	Spokane Valley	Montana	44030	(340) 414-3312	et@Donectempusl	M	06/20/1969	3	0	691
8	8	Mr.	A	Melodie	Simpson	P.O. Box 447, 3081 Nec, St.	Fremont	Alaska	50649	(512) 941-7119	in.consequat.enim	F	08/19/1974	17	1	519
9	9		C	Beau	Bentley	P.O. Box 536, 7781 Nec, Av.	Clairton	Louisiana	89143	(636) 217-2147	vel.venenatis.vel	F	10/18/1973	1	1	670
10	10	Ms.	D	Raphael	Holcomb	Ap #839-7056 Feugiat Avenue	Mesquite	AK	40719	(198) 876-3462	euismod.urna@ne	M	04/25/1974	16	1	679
11	11	Dr.	A	Kessie	Ellis	Ap #976-1288 Risus. Av.	Brown Deer	New Jersey	56599	(496) 365-7412	lacus@purusNulla	F	08/28/1960	8	0	682
12	12	Dr.	E	Deirdre	Cook	Ap #884-2367 Dignissim Avenue	Ventura	Alabama	9580	(964) 973-4389	Duis.a.mi@neccur	M	05/24/1970	16	0	638
13	13		B	August	Kemp	6963 Ultrices, Rd.	Miami Gardens	MO	57851	(584) 105-7449	dolor.Fusce@phar	F	01/25/1963	5	0	592
14	14	Mr.	A	Sierra	Woodward	679-5118 Ipsum Rd.	Watertown	Nevada	61938	(463) 497-8909	amet@InloremDon	M	03/31/1987	16	0	677
15	15	Mr.	E	Kendall	Valentine	5541 Ac Road	Hoboken	MA	11943	(526) 360-1669	Aliquam@Nunc.c	M	01/12/1978	3	1	688
16	16	Dr.	C	Nathaniel	Shepard	Ap #710-8711 Lorem Av.	Norfolk	AK	77986	(836) 106-5944	tristique@augueid.	F	05/04/1976	17	0	544
17	17	Mrs.	B	Aimee	Floyd	588-9628 Tincidunt Rd.	Moline	New Hampshire	9397	(663) 995-0078	vel.lectus.Cum@f	F	08/19/1977	4	0	552
18	18	Dr.	C	Dieter	Gonzalez	450-1948 Ridiculus Street	Evanston	Indiana	69939	(262) 965-7515	massa@egetdictu	F	01/05/1978	4	1	523
19	19	Mr.	E	Daphne	Nunez	P.O. Box 459, 4758 Sed Avenue	Huntington Beach	LA	55528	(722) 273-2247	Nam.ac.nulla@Eu	F	04/25/1967	5	1	594
20	20	Dr.	B	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	M	[illegible]	5	0	[illegible]

Figure 4. Sample Customer Records

For demonstration purposes, the following SAS program is used. You will need to change the program accordingly to retrieve data from other data sources.

The program below requires the following parameter to work correctly:

- `_custID` = Customer ID

```
/*
        Returns the data for a given table
*/
%macro getCustomerDetails;
        /* filter the customer data set by given customer ID */
        data data;
                set temp.customers;
                where custID = &_custID;

                custComments = "";
                custPromotion = "";
                custPromo1ID = "";
                custPromo1Desc = "";
                custPromo2ID = "";
                custPromo2Desc = "";
        run;
%mend;

/* basic parameter validation */
%macro validateParameters;

        %if not %symexist(_custID) %then %do;
                %global _custID;
                %let _custID=;
        %end;

        %if not %symexist(_custPhone) %then %do;
                %global _custPhone;
                %let _custPhone=;
        %end;

        %else %if not %symexist(_custID) or not %symexist(_custPhone) %then %do;
                data data;
                  error = "1";
                  message = "Parameter '_custID' or '_custPhone' is missing.";
                run;
        %end; %else %do;
                %checkds(temp.customers);
                %getCustomerDetails;
        %end;

%mend;
%validateParameters;
```

```
/* write out XML stream */
libname _WEBOUT xml;

data _WEBOUT.CALL_CENTER_WEB_SERVICE_OUTPUT;
  set data;
run;
```

Figure 5. Customer Detail Record Service

The SAS program needs to be registered as a stored process so it can be deployed as a Web service. This is done via SAS® Management Console. The documentation for SAS Management Console shows the process for registering the SAS program as a stored process. There are two important steps to deploy a SAS Stored Process as a Web service:

- specifying the **XMLA Web Service** keyword (Figure 6. General Properties)
- filling in the dialog boxes to define the parameters (Figure 7. Parameters)

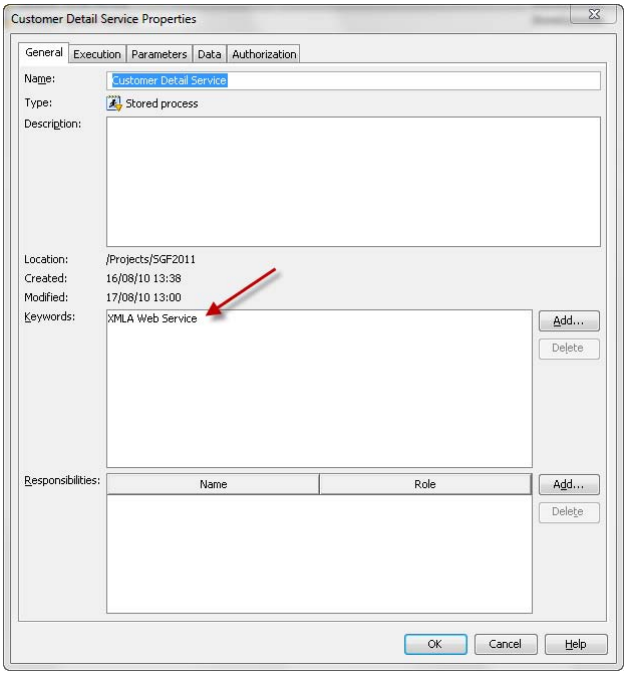


Figure 6. General Properties

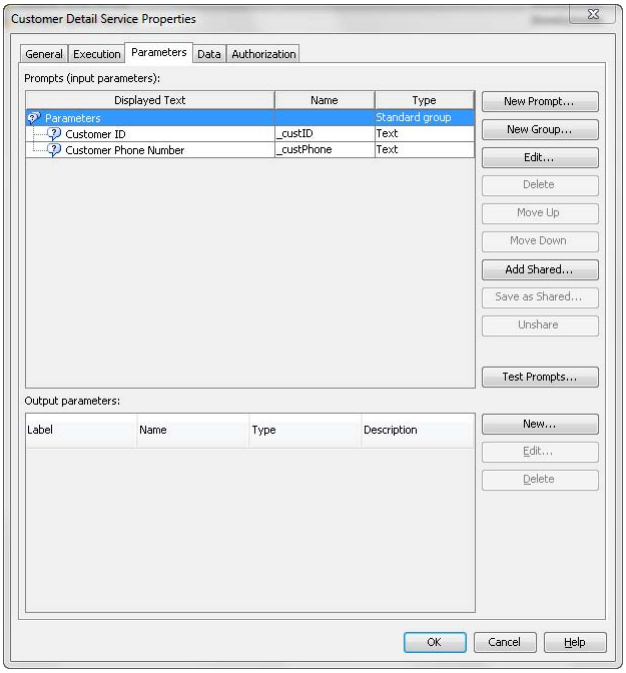


Figure 7. Parameters

Once the SAS program has been registered as a stored process it can be deployed as a Web service. This is done using the SAS Web Service wizard from SAS Management Console. Access this wizard by right-clicking the stored process and selecting **Deploy As Web Service** from the context menu:

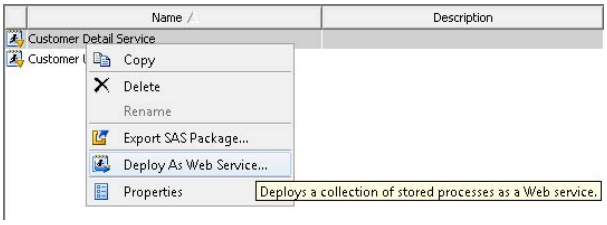


Figure 8. Access 'Deploy As Web Service' Wizard

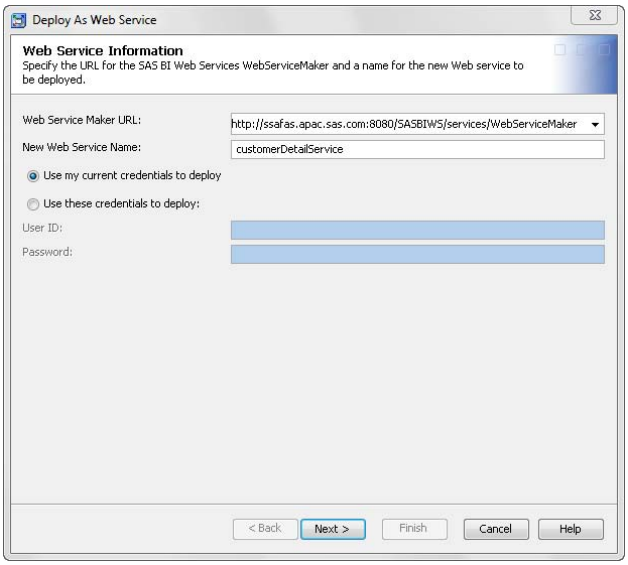


Figure 9. Web Service Information

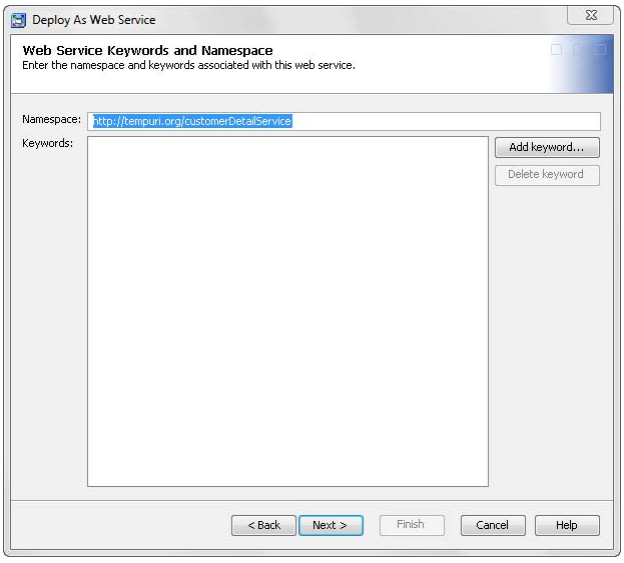


Figure 10. Web Service Keywords

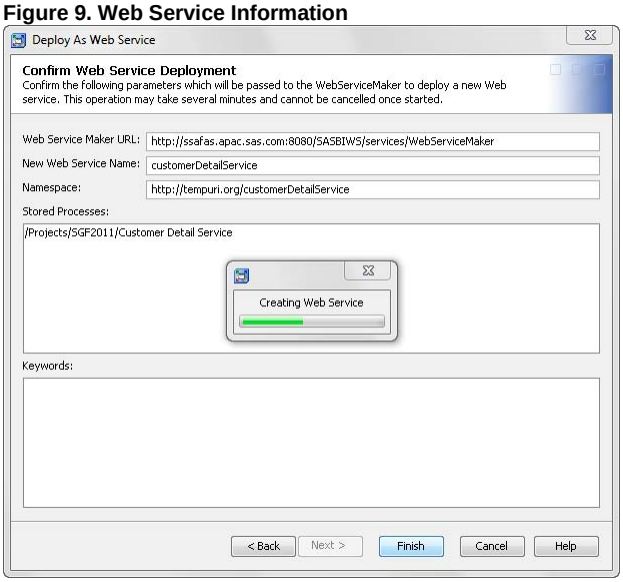


Figure 11. Web Service Deployment

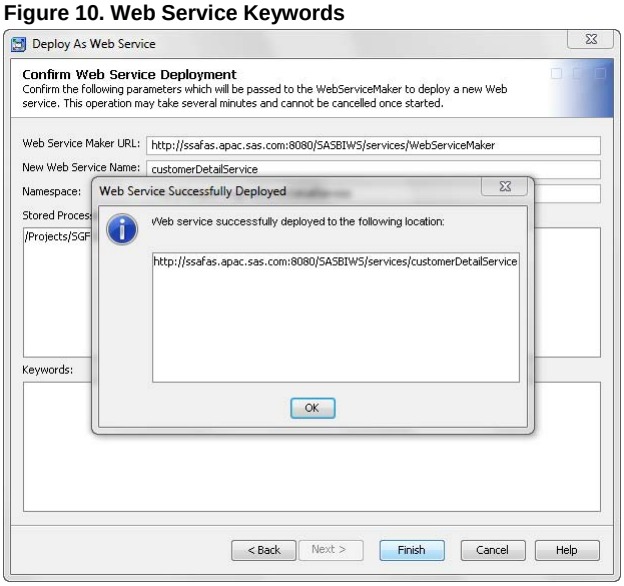


Figure 12. Web Service Deployed

After the Web service is deployed to the application server a confirmation dialog box will be displayed with the endpoint URL for the newly created service (Figure 12. Web Service Deployed).

The Web Service Definition Language (WSDL) file can be accessed by adding "`?wsdl`" to the endpoint URL (Figure 13. WSDL File).

Hint: To copy this URL from the dialog box, use Ctrl-C from the keyboard.

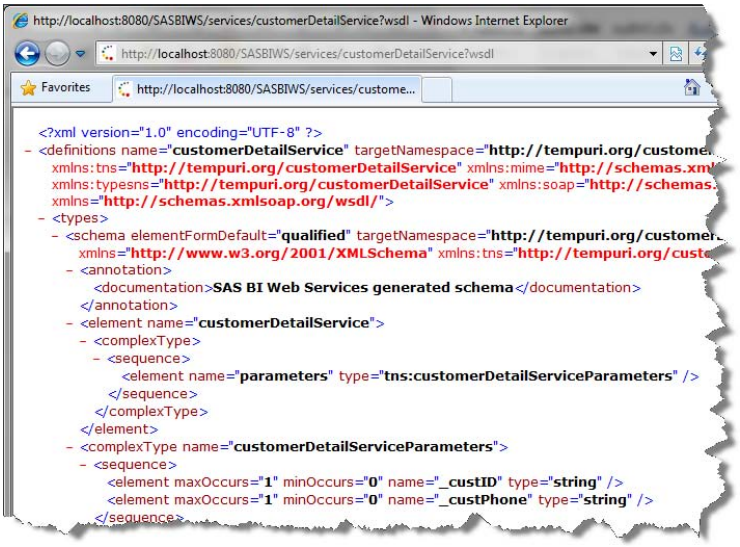


Figure 13. WSDL File

CUSTOMER UPDATE SERVICE

This service will be used to update the customer detail record. It should be capable of retrieving new customer details, which then get updated in the underlying database. This service will also rescore the likelihood of churn based on the new details received. Finally, this service will provide recommended promotions, which can be offered to a customer to prevent churn.

The program below requires the following parameters to work correctly:

- `_custID` = Customer ID
- `_custPhone, _custTitle, _custSegment, _custFirstName, _custLastName, _custAddress1, _custCity, _custState, _custPostcode, _custPhone, _custEmail, _custGender, _custDoB, _custTariff`

```
/*
        Returns the data for a given table
*/
%macro updateCustomerDetails;
        /* update customer data set with new values */
        proc sql;
                update temp.customers
                set custTitle = "&_custTitle",
                        custSegment = "&_custSegment",
                        custFirstName = "&_custFirstName",
                        custLastName = "&_custLastName",
                        custAddress1 = "&_custAddress1",
                        custCity = "&_custCity",
                        custState = "&_custState",
                        custPostcode = &_custPostcode,
                        custPhone = "&_custPhone",
                        custEmail = "&_custEmail",
                        custGender = trim("&_custGender"),
                        custDoB = input("&_custDoB", ddmmyy10.),
                        custTariff = &_custTariff
                where custID = &_custID;
        quit;

        /* re-calculate the churn score */
        data data;
                set temp.customers;
                where custID = &_custID;
```

```
                /* Replace churn score calculation with real SAS Enterprise Miner model: */
                custProb = ranuni(-1) * 100;

                if (custProb > 80) then do;
                    custPromo1ID="TAR001"; custPromo1Desc="Any tariff Upgrade free for 1 month";
                    custPromo2ID="FAM003"; custPromo2Desc="Family and Friends trial package free
for 3 months";
                end;
                else if (custProb > 60) then do;
                    custPromo1ID="FAM002"; custPromo1Desc="Family and Friends trial package free
for 2 months";
                    custPromo2ID="MIN001"; custPromo2Desc="100 free minutes call time to any
network each month";
                end;
                else do;
                    custPromo1ID="MIN002"; custPromo1Desc="200 free minutes call time to any
network each month";
                    custPromo2ID="HST004"; custPromo2Desc="Free handset upgrade and 500MB free
data per month";
                end;
        run;
%mend;

/* some basic parameter validation */
%macro validateParameters;

        %if not %symexist(_custID) %then %do;
                %global _custID;
                %let _custID=;
        %end;

        %if not %symexist(_custPhone) %then %do;
                %global _custPhone;
                %let _custPhone=;
        %end;

        %else %if not %symexist(_custID) or not %symexist(_custPhone) %then %do;
                data data;
                  error = "1";
                  message = "Parameter '_custID' or '_custPhone' is missing.";
                run;
        %end; %else %do;
                %updateCustomerDetails;
        %end;

%mend;
%validateParameters;

/* write out XML stream */
libname _WEBOUT xml;

data _WEBOUT.CALL_CENTER_WEB_SERVICE_OUTPUT;
  set data;
run;
```

Register this SAS program following the same steps described for the “Customer Detail Service”. The only difference is the service name (Figure 14. Customer Update Service) and parameters used (Figure 15. Parameters).

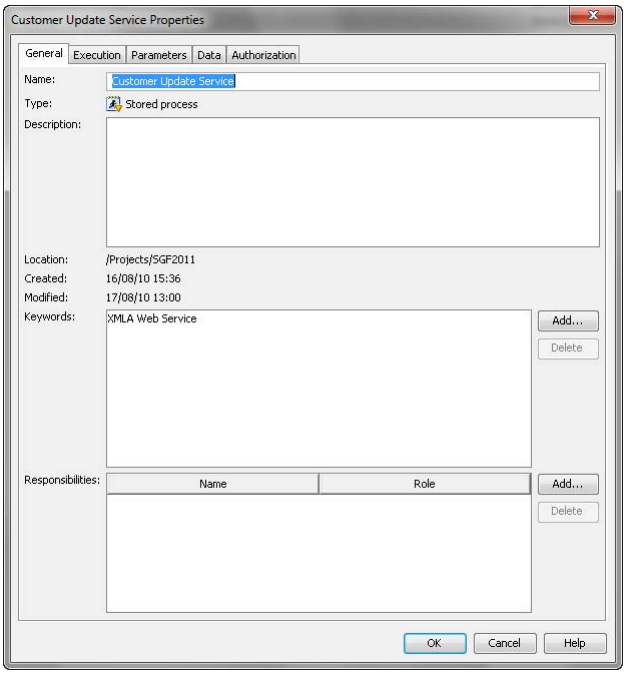


Figure 14. Customer Update Service

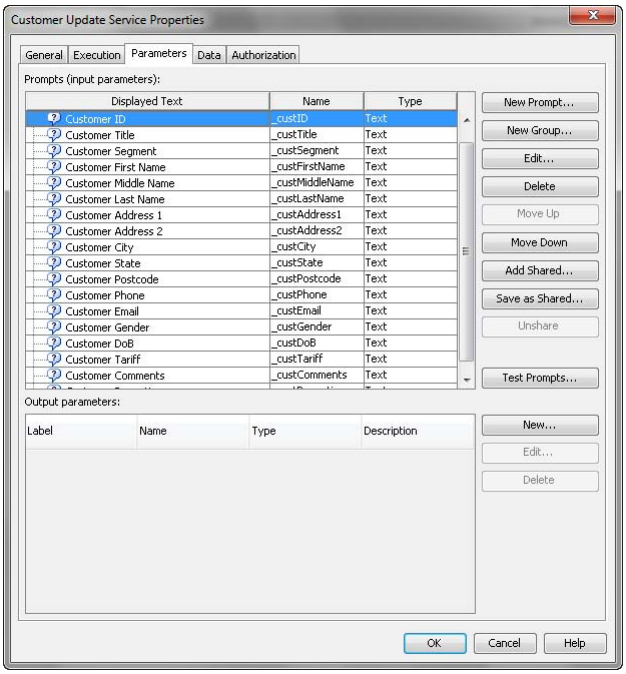


Figure 15. Parameters

BUILDING THE CALL CENTER INTERFACE

The technology used to build the user interface is Adobe Flex. *Rich Internet Applications* (RIA) are Web applications that have the usability of desktop software and give users a more interactive and enriching experience. They are deployable on all major browsers and operating systems, which removes the hassle of performing a client installation. Most systems that run an RIA have Adobe Flash Player installed, which is a prerequisite to running a Flex application.

Note: A zip file with code examples and the full application source code from this paper will also be available at http://www.sascommunity.org/wiki/Building_Enterprise_Applications_using_SAS_real-time_services

The following diagram visualizes the Web service process.

Figure 16. Overview of the Web Service Process

The Adobe Flex client application communicates directly with the middle-tier Java code, making a standardized call (SOAP) to the Web service. The middle tier then makes an Integrated Object Model (IOM) call to the SAS Application Server and executes the stored process. The results are streamed back to the middle tier and are finally returned to the client application.

CALLING A WEB SERVICE IN FLEX

Flex consists of two languages: MXML and ActionScript. MXML is an XML-based language, which incorporates many built-in functions and is generally used to lay out the user interface. When compiled, each MXML tag is generated into ActionScript. ActionScript is a powerful object-oriented programming language. This section of the paper provides a brief overview of the important parts of the code that call the previously created Web service. A basic knowledge of Flex programming is required for understanding the content of this section.

To call your Web service, use the `<mx:WebService>` tag. This tag requires the WSDL address (`http://localhost:8080/SASBIWS/services/customerDetailService?wsdl`), the operation name (`customerDetailService`), and any parameters (`_custID`, `_custPhone`) that you want to pass into the Web service.

Here is an example of how your Web service call should be set up in Flex:

```
<mx:WebService id="webService" showBusyCursor="true"
               wsdl="http://localhost:8080/SASBIWS/services/customerDetailService?wsdl">
       <mx:operation name="customerDetailService"
                 resultFormat="object"
                 result="onCustomerRecordRecieved(event);"
                 fault="getFault(event);">
               <mx:request>
                       <parameters>
                               <_custID> {custID.toString()}</_custID>
                               <_custPhone> {custPhone.toString()}</_custPhone>
                       </parameters>
               </mx:request>
       </mx:operation>
</mx:WebService>
```

The only fields that are unfamiliar at this point should be in the `<mx:operation name="">` tag, specifically `resultFormat`, `result`, and `fault`. The `resultFormat` function lets the compiler know that the return type of the Web service will be an object. The `result` function executes when the operation is completed, and the `fault` function executes if any problems are encountered. Instead of specifying fix values for your parameters (for example, `<_custID>12</_custID>`), you can use ActionScript to assign dynamic values using the following syntax: `<_custID> {custID.toString()}</_custID>`.

To call a Web service, the application uses a `sendWSRequest` function. The Web service gets executed by calling the `send()` function on the operation object as shown in the following code:

```
private function sendWSRequest(service:WebService, reqObj:Object):void {
       var operation:Operation = service.operations[0];
       // add the request object to the operation. The request object contains call parameters
       operation.request = reqObj;
       operation.send();
}
```

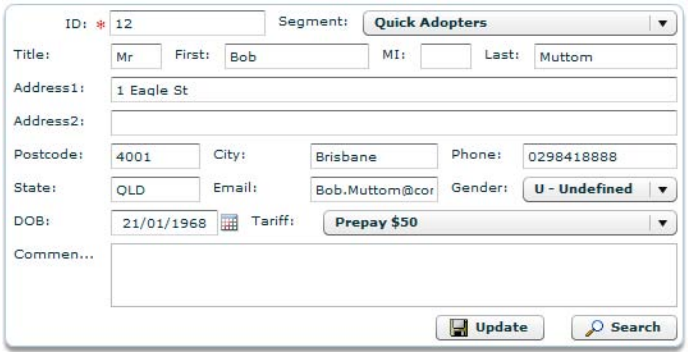
The `onCustomerRecordRecieved` function retrieves the results that are generated by the Web service and places them into an ArrayCollection:

```
private function onCustomerRecordRecieved(evt:ResultEvent):void {

       //retrieve the SAS table name from the XML property file
       var drWS:XMLList = xmlProperties..service.(@name == "DETAILRECORD");
       var tableName:String = String(drWS[0].@outputTable);
       // do we have a result stream
       if (evt.result.Streams) {
               // create a new array collection using the ArrayUtil class
               var orArray:ArrayCollection = new
ArrayCollection(ArrayUtil.toArray(evt.result.Streams._WEBOUT.Value.TABLE[tableName].source));
               if (orArray.length == 0)
                       orArray = new
ArrayCollection(ArrayUtil.toArray(evt.result.Streams._WEBOUT.Value.TABLE[tableName]));
               // if the array contains values use the field mappings to assign values
               if (orArray.length > 0) {
                       var firstRow:Object = orArray[0];
                       var fieldMappings:XMLList = drWS..field;
                       loadFields(firstRow, fieldMappings);
               }else{
                       Alert.show("Customer ID " + this.custID.text + " not found.");
               }
       }else{
                       Alert.show("Customer ID " + this.custID.text + " not found.");
       }
}
```

The preceding code snippet reads the SAS output table name from a property file, which configures the application. This way you do not need to re-compile your application if the Web service output table changes. The application also calls the `loadFields` function, which reads each customer record detail and assigns it to an input field.

Once all fields are loaded, the user interface shows the customer details:



The complete Flex code can be downloaded at http://www.sascommunity.org/wiki/Building_Enterprise_Applications_using_SAS_real-time_services. This code also produces an interactive bar chart, line plot, and other call center interface-related outputs. Figure 17 shows the final call center application interface.

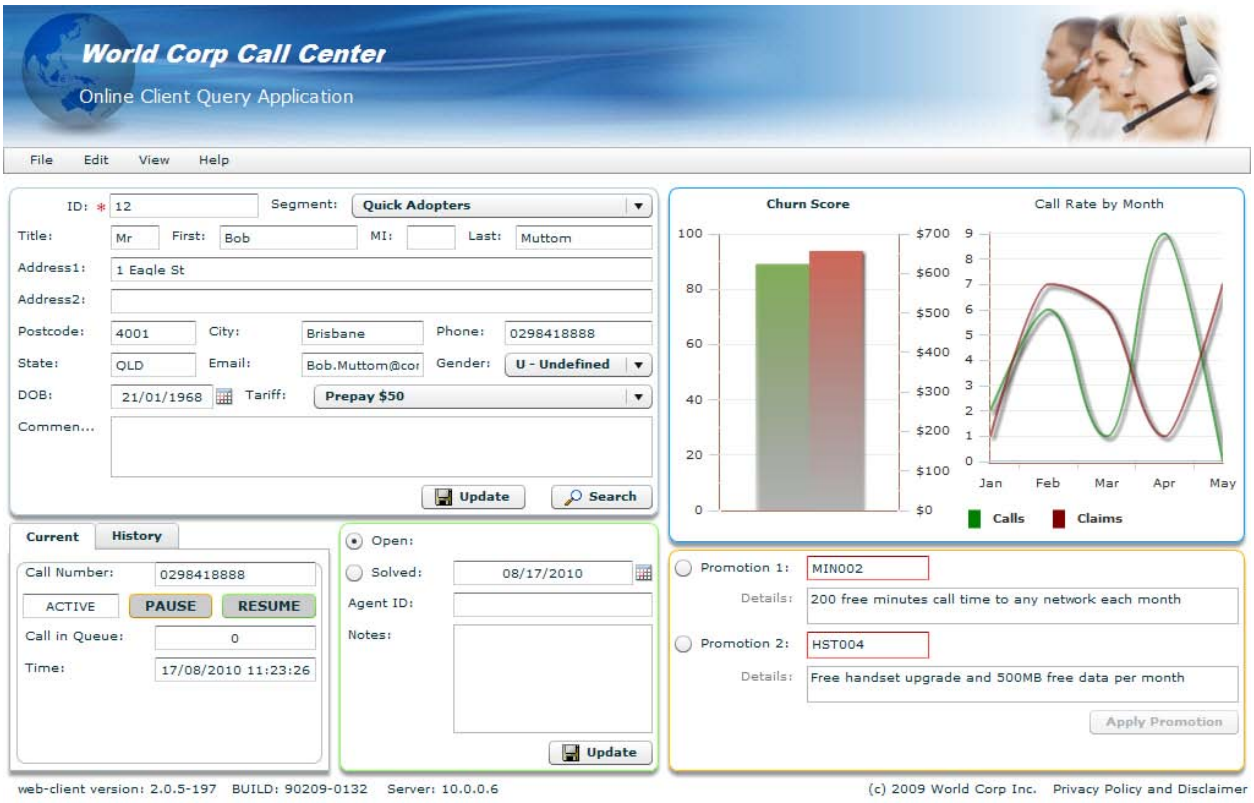


Figure 17. Final Call Center Application Interface

The application also provides the user with the capability to update the customer record. The updated fields can be written back to the server by pressing the Update button. This will initiate a request to execute the '`customerUpdateService`' Web service, which updates the SAS data set.

Field values are passed to the SAS Web service using a request object (`reqObj`):

```
private function updateCustByID(event:Event):void {

        var reqObj:Object = new Object();
        reqObj.parameters = {
                _custID: this.custID.text.toString(),
                _custTitle: this.custTitle.text.toString(),
                _custSegment: String(this.custSegment.selectedItem.@name),
                _custFirstName: this.custFirstName.text.toString(),
                _custMiddleName: this.custMiddleName.text.toString(),
                _custLastName: this.custLastName.text.toString(),
                _custAddress1: this.custAddress1.text.toString(),
                _custAddress2: this.custAddress2.text.toString(),
                _custCity: this.custCity.text.toString(),
                _custState: this.custState.text.toString(),
                _custPostcode: this.custPostcode.text.toString(),
                _custPhone: this.custPhone.text.toString(),
                _custEmail: this.custEmail.text.toString(),
                _custGender: this.custGender.selectedItem.toString(),
                _custDoB: dfShort.format(this.custDOB.selectedDate),
                _custTariff: String(this.custTariff.selectedItem.@name),
                _custComments: this.custComments.text.toString(),
                _custPromotion: this.custPromotion.text.toString()
        };
        // use our utility class to send the web service request
        sendWSRequest(urService, reqObj);
}
```

This service will re-calculate the recommended promotion based on the new information. For the purpose of this paper, the customer churn probability is randomly generated (that is, replace this with a SAS® Enterprise Miner™ scoring model). Based on the churn probability, the new promotion offers are calculated. This prototype application uses a simplified SAS code that needs to be updated in a production environment. See the "Customer Update Service" above for more details.

GLOSSARY

- WSDL – The **Web Services Description Language** (WSDL) is an XML-based language that provides a model for describing Web services. (http://en.wikipedia.org/wiki/Web_Services_Description_Language)

- SOA – **Service-oriented architecture** (SOA) is a flexible set of design principles used during the phases of systems development and integration in computing. A system based on an SOA architecture will package functionality as a suite of interoperable services that can be used within multiple separate systems from several business domains. (http://en.wikipedia.org/wiki/Service-oriented_architecture)

- SOAP – SOAP, originally defined as **Simple Object Access Protocol**, is a protocol specification for exchanging structured information in the implementation of Web services in computer networks. It relies on Extensible Markup Language (XML) for its message format, and usually relies on other Application Layer protocols, most notably Remote Procedure Call (RPC) and Hypertext Transfer Protocol (HTTP), for message negotiation and transmission. (http://en.wikipedia.org/wiki/SOAP)

- IOM – The **Integrated Object Model** (IOM) in SAS Integration Technologies provides distributed object interfaces to SAS software features. To call these interfaces, clients can use industry-standard languages, programming tools, and communication protocols. The interfaces are built into SAS and are available to clients whenever SAS is executed as an object server. (http://support.sas.com/documentation/onlinedoc/inttech)

CONCLUSION

Using SAS BI Web service is an easy and convenient way to stream results from SAS analytics processes into enterprise applications. Real-time services allow you to leverage the power of SAS across the enterprise and beyond, while integrating the processes in your SOA.

REFERENCES

Flynn, Joe. 2010. “Flex Your SAS® Muscle.” *Proceedings of the SAS Global Forum 2010 Conference.* Cary, NC: SAS Institute Inc.

Vincent, Stephen A. 2010. “SAS® Application Messaging: How to Integrate Disparate Processes in Your Service-Oriented Architecture.” *Proceedings of the SAS Global Forum 2010 Conference.* Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

I would like to thank Jeremy Rankcom for his help in reviewing this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name:	Falko Schulz
Enterprise:	SAS Institute Inc., Australia
Address:	1 Eagle St
City, State ZIP:	Brisbane, QLD, 4001
Work Phone:	07-3233 320
E-mail:	Falko.Schulz@sas.com
Web:	http://www.sas.com