TIP/ix Support for Web Services & SOAP

For TIP/ix version 2015/01/01 2.5 R0 – 0288 or later

Documentation: January 2015

Table of Contents

Introduction 3
TIP/ix directories used for Web Services 3
Transactional vs Conversational 4
Additional tipix.conf parameters 5
session.conf parameters for SOAP interface 7
Programming a Web Service 7
XML – COBOL data mapping 8
`tipcblxml` utility 10
Application Program Calls 11
TIPMSGI for SOAP 11
TIPMSGO for SOAP 11
TIPRTN for SOAP 11
TIPMSGE for SOAP 11
TIPTERM for SOAP 12
TIPSUSPEND Suspend/resume a transaction 12
FIRSTXMLTAG 13
TIPXML 14
Invoking an outside Web Service 15
GETWEBSERVICE 15
A tutorial using TSP 17
Tutorial Message Exchange 20
AJAX Invoking TIP/ix Web Service 23
C# Invoking TSP Web Service 25
Soap Template files 28
Client programming considerations 30
TSP conversational Example 31

Introduction

The purpose of this document is to describe features of TIP/ix which provide support for Web Services. A few specific APIs have been added to TIP/ix so that application programs can invoke other Web Services and also so that a TIP/ix COBOL application program could act as a Web Service.

The basis of Web Services is to exchange data using SOAP (Simple Object Access Protocol) and having the data encoded in XML. SOAP messages are most often layered on HTTP and this is the method which TIP/ix supports.

XML is not commonly used in COBOL applications so there is a need for converting regular COBOL style data to and from XML messages. One possible way to do this is by using a feature of the recent Micro Focus COBOL compiler that lets you read/write a buffer of XML data. TIP/ix also has a API called TIPXML for doing this data conversion as well.

The client side of a 'web service' is often a JavaScript using AJAX (Asynchronous JavaScript and XML). JavaScript can send/receive data in XML format but it also often uses a data format called JSON (JavaScript Object Notation).

Most TIP/ix applications use the TIPMSGO/TIPMSGI calls to interface with the end user's terminal. To make is easier for existing TIP/ix applications to be adapted to act as a web service TIPMSGO/TIPMSGI have been extended to convert the MCS-DATA to/from XML or JSON data.

Applications originally developed on the Unisys 2200 for DPS/2200 would be using D$SEND/D$READ which is equivalent in functionality to TIPMSGO/TIPMSGI.

TIP/ix directories used for Web Services

Under the $TIPROOT directory where you have installed TIP/ix there are now a few extra directories. The `tipsoa` directory has a few starter SOAP template files.

`tipsoa`		Holds various ***template files*** used by the system. (SOA stands for Service Oriented Architecture)
	default.reply	A default SOAP/XML template file used for sending reply back to a Web Service client. The line holding ${reply} will be replaced by the XML text from TIPTERM T-PUT or TIPMSGO.
	default.error	A default SOAP.XML template file used for sending an error back to a Web Service client The line holding ${text} will be replaced by the error message text.
`cpy`		Hold COBOL copy books generated by the tipcblxml utility. You may want to add this directory to the COBCPY environment variable which the Micro Focus COBOL compiler uses to search for COPY books.

Transactional vs Conversational

The first support of Web Services in TIP/ix was strictly transactional. One input message, processing, one reply message and then the COBOL program was expected to terminate and the TIP/ix user interface process would close the TCP/IP socket and exit. As of version '2014/02/23 2.5 R0 - 0264' of TIP/ix the default now is that web services are conversational much like normal TIP/ix application programs. If you want TIP/ix to revert back to the original 'transactional mode' then you must add the following to the tipix.conf file in the $TIPROOT/conf directory:

```
PARAM SOAPCONVersational=No
```

The default for this parameter is now YES. Even with 'transactional mode' the TIP/ix interface process will linger after the COBOL program terminates and if another SOAP message arrives on the socket, it will process it like a new request. By keeping the TCP/IP socket open performance is improved somewhat. If you want the socket to be closed immediately after the program terminates then add the following to the tipix.conf file in $TIPROOT/conf directory.

```
PARAM SOAPLINGER=NO
```

Optionally, you may define how many seconds TIP/ix should linger with the socket open, waiting for the next request and the default is 10 seconds. Example:

```
PARAM SOAPLINGER=30
```

When running in 'transaction mode' the COBOL program will see in the PIB the 88 level items with the value of PIB-TIPWEBSERVICE set to TRUE and also PIB-TYPE-WEBSERVICE is set to TRUE.

When running in 'conversation mode' the COBOL program will see the 88 level items PIB-TIPWEBSERVICECONV and PIB-TYPE-SOAPCONV set to TRUE.

Note that HTTP is a very request/response style of protocol. In order to implement a conversation there needs to be some sort of session identifier used in the messages sent back and forth. The session id is sent back in the HTTP header as a 'cookie' using the Set-Cookie directive. The name of the 'cookie' is by default 'Session_Id'. So the line in the HTTP header might look like the following:

```
Set-Cookie: Session_Id=EsMFAAAAAEqe; expires=Fri, 14-Feb-2014 22:01:59 GMT; Path=/
```

The expiry time is based on the value of PARAM WEBTIMEOUT defined in the `tipix.conf` file. The session-id value is a unique value expressed in Base64 notation and the **value is unique for each interaction** (or exchange of messages) so whenever the client side gets a session-id, it **must** save the value and **return it on the next message sent** to TIP/ix.

The session-id may be sent to TIP/ix as a cookie in the HTTP header as follows:

```
Cookie: Session_Id=EsMFAAAAAEqe
```

If using Cookies is not possible, then TIP/ix may send and receive the session-id as a known XML or JSON data item within the body of the message.

Additional tipix.conf parameters

Parameters which may be in the TIP/ix configuration file (limit 23 bytes each):

SOAPCOMPUTERREQUIRED=Y	'Yes' indicates that a valid computer name define via smterm must exist and be available or the login is rejected. Default is `No`
SOAPCOMPUTERSESSIONTAG=	Name to indicate the session number (i.e. instance/occurrence) from the same computer/user. Default is `Computer_Session_Num`
SOAPCOMPUTERTAG=name	the name used to indicate the user's (client side) computer name. This value is used lookup a 'terminal name' inside TIP/ix. Default is `Computer_Name`
SOAPDOMAINTAG=name	name indicates the network domain for login. Default is `Domain_Name`
SOAPERRORTAG=name	Name used for TIPMSGEO text. Default is `Error_Data`
SOAPFUNCTIONkeyTAG=name	name used to hold a function key value within the data message sent from the client. Default is `Function_Key`
SOAPLOGIN=Yes/No	Default is NO meaning that web service clients do not need to log into Tip/ix. For YES the initial SOAP message must contain the user login information with userid & password and optionally a domain name and computer name.
SOAPMCSFUNCTIONTAG=name	the name used to hold the type of MCS function being sent to the client. * Default is `Mcs_Function` The value may be one of msgo, msgov, title, menu, list, etc..
SOAPMCSNAME=name	the name used to hold the MCS format name used by the application. * Default is `Mcs_Name`
SOAPMCSNAMETAG=name	Tag name used to identify the most recent value used in MCS-NAME. Default is `Mcs_Name`
SOAPPASSWDTAG=name	name to indicate the user's password for login. Default is `Password`

SOAPSESSIONTAG=name	Tag name used to identify the session-id value. The TIP/ix session id value is returned on each output message and changes so the client side must save it and send back the most recent value received. The default tag is: `MySession_Id`
SOAPTEXTTAG=name	name used for additional data for the functions such as TIPTITLE, TIPMENU, ROLL, etc. Default is `Text_Data`
SOAPTIPLISTTAG=name	Name used to define TIPLIST data. Subitem name for each line is 'item'. Default is `Tiplist_Data`
SOAPTRIDTAG=name	name used to hold the application transaction id. * Default is `Transaction_Id`
SOAPTRIDTAG=name	Tag name used to identify the current transaction-id name. Default is `Transaction_Id`
SOAPUSERIDTAG=name	name used to indicate the user-id for login. Default is `User_Id`
WEBSERVICE=ports	List ports to be listened to for SOAP. If more than one port separate by commas. Additional control information may be defined in the session.conf file in $TIPROOT/conf

* When in conversation mode these data items are included with the MCS-DATA fields sent to the client.

session.conf parameters for SOAP interface

The session.conf file is used to define various options for different kinds of interfaces to TIP/ix. For SOAP you can declare a TCP/IP port number which was defined by the WEBSERVICE parameter in tipix.conf and then declare extra conditions for that port. For example if tipix.conf had WEBSERVICE=8081,8082 then the session.conf might contain:

```
[8081]
LOGIN=NO
CONVERSATION=NO
[8082]
LOGIN=YES
CONVERSATION=YES
```

The above defines that SOAP connections for port 8081 do not require login and they run in 'transaction mode' while SOAP connections to port 8082 do require user login and they run in 'conversation mode'.

Programming a Web Service

This section describes how TIP/ix COBOL application programs are able to act as a web service.

TIP/ix already has a component for distributed transaction processing which listens for TCP/IP connections from other TIP/ix systems (tipdtp). This same component also listens for TIP/ws direct connect requests. The 'tipdtp' process will also listen on a defined TCP/IP port for Web Service connections. The definition of which TCP/IP port to listen is declared in the `tipix.conf` file as a parameter statement as follows:

```
PARAM WEBSERVICE=8082
```

The value of this parameter is the TCP/IP port that is to be listened on for a connection. It is possible to specify more than one port for WEBSERVICE. For example:

```
PARAM WEBSERVICE=8080,8082
```

Declares that ports 8080 and 8082 are both for the WebService interface (aka SOAP).

The default buffer for passing XML data is set to a size of 32K. You may define other sizes with a maximum of 64K with the following parameter in `tipix.conf`.

```
PARAM WEBMCSBUFSIZE=24000
```

When a connection is received on the defined TCP/IP port, it will be assumed to be a 'web service' message in SOAP/HTTP format. That last part of the `SOAPAction` will be used as the TIP/ix transaction code. Since TIP/ix transactions are limited to 8 bytes the `SOAPAction` will be truncated at 8 characters and converted to upper case to locate the transaction program that will service this request. For example if the `SOAPAction` is as follows:

```
SOAPAction: "http://tempuri.org/CBMMugShotWebService/TokMugShot/GetMugshot"
```

Then the TIP/ix transaction code GETMUGSH would be used.

The COBOL application program will know it is running as a web service because the PIB-TERM-TYPE will have a value of PIB-TIPWEBSERVICE and PIB-TYPE will also have a value of PIB-TYPE-WEBSERVICE.

Once the transaction program is given control, it may read the input data (in any number of ways), process the transaction request and then reply in a number of ways.

One way is via the TIPMSGO/TIPMSGI interface. When running as a web service, TIPMSGI will convert the input data (XML or JSON) into the MCS-DATA area using the provided data mapping rules. The data mapping rules used for TIPMSGI are found by using the MCS-NAME field and searching for that mapping rules file. Likewise, a TIPMSGO will convert the MCS-DATA area back into an XML or JSON message using the provided data mapping rules. TIPMSGE will take the first 80 characters of the error message and pass that back as a SOAP reply with the text tagged as <McsError>. Extended MCS functions like TIPMSGEO, FCC-MODS, CURSOR-MODS etc are not be supported when operating as a web service and they are just ignored.

Another way for the application to handle XML data is through the API provided for this purpose (TIPXML).

XML – COBOL data mapping

For a TIPMSGI, the MCS-NAME will be used to search for the data mapping rules and XML template to be processing. The file will be searched for in $TIPROOT/tipsoa and then in /etc/default. The file name will be <MCS-NAME>.mapxml.

So if the MCS-NAME field contains ‘MYDTA’ then the file name searched for is MYDTA.xmlcob. Let’s look at an example. If the MCS-DATA is as follows:

```
05  MENU-FUNCTION      PICTURE X(2).
05  MENU-KEY           PICTURE X(8).
```

Then the XML mapping rules might look like:

```
05  MENU-FUNCTION      PICTURE X(2) IDENTIFIED BY "FUNC".
05  MENU-KEY           PICTURE X(8) IDENTIFIED BY "KEYIS".
```

The actual XML input data might look like the following:

```
<FUNC>AD</FUNC>
<KEYIS>RBW00101</KEYIS>
```

The syntax used here is based on that used by Micro Focus COBOL XML pre-processor. Refer to the latest Micro Focus online documentation and then look under COBOL Language → Additional Topics → XML Syntax for details.

A more detailed example of this XML-COBOL syntax follows:

```
       02  library            identified by "library".
           05  libname   PIC X(40)   identified by "library-name".
           05  Comment   PIC X(25).
           05  book             identified by "book"
                               occurs 10 times
                               count in library-book-count.
             10  book-title  PIC X(80) identified by "title".
             10  book-author PIC X(80) identified by "author".
             10  book-toc              identified by "toc".
                15  book-toc-section  occurs 15 times
                                 count in book-section-count
                                 identified by "section"
                                 PIC X(20).
             10  book-price PIC S999V99 COMP-3  identified by "Price"
                                          is attribute.
           05  shipper   PIC X(40)   identified by "shipper".
```

For the above data description the actual XML data might look like the following:

```
<library>
<library-name>Mannheim Central Library</library-name>
<book> <price>-1.50</price>
  <title>Book1</title>
  <toc><section>b1 section 1</section>
       <section>b1 section 2</section>
  </toc>
</book>
<book price="9.75">
  <title>Book2</title>
  <toc><section>b2 section 1</section>
       <section>b2 section 2</section>
       <section>b2 section 3</section>
  </toc>
  </book>
</library>
```

Note the field ‘Comment’ does not have any IDENTIFIED clause so this field is in the COBOL record layout but it does not appear in the XML data.

Also note that ‘Price’ is defined as an ATTRIBUTE of ‘Book’. However, it will be placed into the correct COBOL record location if it is an attribute or a subordinate XML tag to ‘Book’.

A name defined by COUNT IN is generated into the simple COBOL copy book as a PICTURE 9 and the application prorgam would put into that field the actual number of occurences of the data item. At run-time the value from the COUNT IN field is used to determine how many occurences of the field (or group of fields) is generated. When converting XML into COBOL record the COUNT IN field will be set to the number of occurences of the data item processed.

`tipcblxml` utility

This utility program will take a COBOL COPY book with the additional XML syntax and from that generate a regular COBOL COPY book that could be copied/compiled directly into a program and a data mapping control file that can be used at run-time by TIP/ix. The usage of this utility is:

```
TIP/ix COBOL - XML processor, Version  1.22 2013/01/11
Proper Command format follows

 tipcblxml  [options] copybook-with-xml

Where [options] are:
   -o mapname    Name of the mapping rules file
   -m mapname    MCS-NAME for the mapping rules file
              Use only one of -o or -m otherwise
              name defaults from the input file name
   -c copyname   Name of the COBOL copy book created
   -I            Do not assume IBMCOMP (ie. COMP-4 SYNC)
   -a            Make all COBOL fields into XML tag
   -e n          Where 'n' is the size used for TIPMSGE (Default: 32)
   -g group      TIP/ix Group Name for the mapping rules file;
                   (Becomes subdirectory under tipsoa)
```

For example:

```
  >tipcblxml lib.cpy
Generated COPY book: LIB & Data Mapping rules file: lib
  >tipcblxml -m 'TF$TSP1_' XM-TSP1X
Generated COPY book: XM-TSP1X & Data Mapping rules file: tf_tsp1x
```

This utility writes the COBOL COPY book out to `$TIPROOT/cpy` directory and the XML mapping rules file is written to `$TIPROOT/tipsoa/name.mapxml`.

If the `-g` *`groupname`* was given then the XML mapping rules is written to `$TIPROOT/tipsoa/`*`groupname`*`/name.mapxml`. (Keep *`groupname`* 8 characters or less.)

Any field defined as **PIC X(n) COMP-X** will be emitted to the simple COPY book as just PIC X(n) but the data for that field will be processed in Base64 format. This takes the binary value of the field and converts it to a string of Base64 when creating XML and vice versa when converting the XML back to a COBOL record format.

Application Program Calls

There are a few new calls for the Web Service interface and some older calls have been made Web Service aware and function differently. At run-time the MCS-DATA may optionally be processed in either XML or JSON format. The method used is determined by the presence of **xml** or **json** in the HTTP header tags **Content-type:** or **Accept:.**

TIPMSGI for SOAP

Once you know what the XML/JSON data tags are going to be and what the data will look like, then you can write a copy book with the COBOL-XML syntax, then process that with `tipcblxml` to generate a simple copy book for use in your COBOL application and the data mapping rules control file for use by TIP/ix at run-time. When the COBOL transaction program gets control as a 'web service' it can simply fill in MCS-NAME with the correct name and then call TIPMSGI. At this point TIPMSGI will know it is running as a web service, locate the data mapping rules and then convert the input XML data into the MCS-DATA area according to the rules it has been given.

At this point your transaction may perform the desired business logic.

TIPMSGO for SOAP

Once the transaction has completed, it should prepare its reply by filling in the MCS-NAME with the correct value for the data mapping rules, fill in the data of MCS-DATA accordingly, set MCS-COUNT as required and then call TIPMSGO. At this point TIPMSGO will know it is running as a web service and convert the MCS-DATA into an XML (or JSON) message based on the given rules and send that back to the client that originally requested the service. A TIPMSGOV is handled exactly the same as TIPMSGO but only format positioning information is ignored.

TIPRTN for SOAP

A web service is based on the request-response model and is not conversational like an end-user terminal session might be. So when the application program has sent its reply back it should terminate. For a Web Service, Ideally you should terminate with EXIT PROGRAM instead of CALL TIPRTN.

If the transaction aborts or fails to send back a reply then a default XML message with the appropriate error message will be sent back to indicate the problem has happened.

TIPMSGE for SOAP

A call TIPMSGE will take up to the 1st 32 bytes of the 'error message' and send it back as a **`Soap:Fault`** message with the text enclosed by `<faultstring>` and `</faultstring>`.

Defining `-e nn` to `tipcblxml` can be used to declare a different error message length to be used.

TIPTERM for SOAP

If you want to read the raw SOAP XML/JSON data you could do this with a TIPTERM T-GET call. When running as a web service TIP/ix will then return the raw XML data into the DCIO-INP-DATA buffer.

Likewise, a TIPTERM T-PUT would take the raw XML data from DCIO-OUT-DATA, prefix it with the SOAP header and send back to the client.

The application program is responsible for correctly forming the XML message. This could be done by using the Micro Focus XML pre-processor XD, READ/WRITE verbs.

In addition TIP/ix will provide some XML conversion routines that will use the data mapping rules processed by the `tipcblxml` utility.

TIPSUSPEND Suspend/resume a transaction

The initial support for Web Services in TIP/ix was strictly 'transactional'. This is now an optional mode of operation. But if running with the strict transactional mode (single input, process, single output), rather than conversational there might be some situations where you would like to avoid committing updates to the data until a later interaction with the web service program.

The API call TIPSUSPEND has been created for this purpose. If a transaction is suspended and not resumed within a defined period of time, then TIP/ix will automatically abort the transaction and rollback all pending data updates. The default time for this is 15 seconds. In `tipix.conf` you can add a parameter to change the default time out for a transaction suspension.

```
PARAM TRANSUSPEND=nn
```

Where 'nn' is the number of seconds to allow. By default there is no CDA data saved with the transactions. Often it makes sense to have some additional data associated with the transaction. To have the program's CDA data saved with the suspended transaction add the following parameter to the tipix.conf file.

```
PARAM TRANSUSPENDCDA=YES
```

Suspending a transaction will return the TIP/ix global transaction identifier to the program. The program should save that someplace for later use. Maybe send it back as part of the reply to the web service client and then on a later web service request the transaction identifier could be sent back in. To suspend the current transaction (assuming some data has been updated), do the following:

```
   WORKING-STORAGE or LINKAGE SECTION.
. . .
   05  TRAN-TIME-OUT    PIC 9(8) BINARY.
   05  TRANS-ID         PIC X(24).
```

Then in the code add:

```
   CALL "TIPSUSPEND" USING FCS-PUT, TRANS-ID, TRAN-TIME-OUT
```

This will put the current transaction into a suspended state for up the number of seconds in TRAN-TIME-OUT. If the TRAN-TIME-OUT parameter is omitted then the system default value is used. If the transaction is not resumed within that time frame it will be assumed to be aborted and the data will be rolled back. The 'transaction Id' is returned in TRANS-ID as a text string and this valued must be used later to resume the transaction. If there had been no data updated then there is no pending transaction so the status of PIB-NOT-HELD is returned.

To re-join a previously suspended transaction do the following:

```
   CALL "TIPSUSPEND" USING FCS-GET, TRANS-ID
```

If the transaction identifier was not found (likely due to some timeout and rollback) then a status of PIB-NOT-FOUND is returned. When the TIPSUSPEND call has been successful the status is set to PIB-GOOD.

FIRSTXMLTAG

This routine will scan a buffer of XML text and return the first XML tag found in upper case. For example:

```
          05  WEB-BUF-RTN  PIC S9(8) COMP-4.
          05  WEB-BUF      PIC X(3000).
          05  WEB-TAG      PIC X(48).

           CALL "FIRSTXMLTAG" USING WEB-TAG,
                                    WEB-BUF,
                                    WEB-BUF-RTN.
```

WEB-BUF-RTN should hold the size of the XML data that is in WEB-BUF. On return WEB-TAG will hold the first XML tag. If no XML was found then WEB-TAG will be all SPACES.

TIPXML

You can convert a simple COBOL data record into XML in a buffer using the following call:

```
       CALL "TIPXML" USING        T-PUT,
                                  XML-MAP-NAME,
                                  COB-REC,
                                  XML-BUF,
                                  XML-LEN.

05  T-PUT          PIC X VALUE "P".
05  T-GET          PIC X VALUE "G".
05  T-TEXT         PIC X VALUE "S".
05  XML-TAG        PIC X(48) VALUE "MCSERROR".
05  XML-MAP-NAME   PIC X(8) VALUE "MYNAME".
05  COB-REC . . . .

05  XML-LEN        PIC 9(8) COMP-4.
05  XML-BUF        PIC X(4000).
```

The TIPXML parameters are as follows:

1	Function code	T-PUT will convert COB-REC into XML data and place it into XML-BUF T-GET will take the XML data from XML-BUF and convert it back to COBOL format in COB-REC T-TEXT will return up to 80 bytes from the XML-BUF from the tag named in parameter 2.
2	Map name	The holds the name of the XML data mapping rules
3	COB-REC	A COBOL record as defined by the XML-MAP-NAME
4	XML-BUF	A field large enough to hold the XML data
5	XML-LEN	PIC 9(8) COMP-4 that holds the length of data in XML-BUF for T-GET function. For T-PUT, this is passed as the maximum length of XML-BUF and is returned as the length of the XML data placed into XML-BUF

Invoking an outside Web Service

This section will describe how an application program running under TIP/ix could send a message to an outside Web Service and get a reply.

GETWEBSERVICE

This routine reads a template file and merges in some application supplied data, sends the message and retrieves the reply. This API may be used from any batch or transaction program. The routine uses the following data:

```
       77  PRM-LEN      PIC 9(4) COMP-4 VALUE 20.
       77  NUM-PRMS     PIC 9(4) COMP-4 VALUE 2.
       77  WEB-BUF-LEN  PIC 9(8) COMP-4 VALUE 50000.
       77  WEB-BUF-RTN  PIC S9(8) COMP-4 VALUE -1.
       01  WEB-BUF      PIC X(50000).
       01  WEB-SERVICE  PIC X(64) VALUE "GetMugShot".
       01  WEB-PARAMS.
           05 WEB-PARAM1  PIC X(20) VALUE "sid='650834'".
           05 WEB-PARAM2  PIC X(20) VALUE "sid='1525680T'".
```

A call may be made as follows:

```
         CALL "GETWEBSERVICE" USING WEB-SERVICE,
                                    NUM-PRMS,
                                    PRM-LEN,
                                    WEB-PARAMS,
                                    WEB-BUF,
                                    WEB-BUF-LEN,
                                    WEB-BUF-RTN.
```

The parameter meaning is as follows:

1	WEB-SERVICE	64 byte field holding the name of the Web Service template file.
2	NUM-PRMS	2 byte binary field holding the number of parameters in the parameter table
3	PRM-LEN	2 byte binary field holding the length of each parameter
4	WEB-PARAMS	A table of parameters
5	WEB-BUF	Area to receive the reply data from the Web Service
6	WEB-BUF-LEN	4 byte binary field holding the length of WEB-BUF
7	WEB-BUF-RTN	Signed 4 byte binary field to receive the length of the XML data returned to the application. -1 indicates the Web Service template file was not found. -2 indicates the host name was missing from the Web Service template file. -3 indicates there was an error opening the connection to the Web Service host computer. -5 indicates that the XML data is longer than the size declared in WEB-BUF-LEN and was truncated. -6 indicates the requested tag was not found.

The Web Service template file is located in `$TIPROOT/tipsoa` directory or `$TIPROOT/conf` directory or in the `/etc/default` directory.

With the example above the template file would be called `GetMugShot.xml`. A sample of such a file follows:

```
# This should be the complete XML POST message to be sent
#   Lines starting with '#' are comments and skipped
#   Lines starting with '~' are concatenated to previous line with no CR LF
#   Lines starting with '%' will have the characters translated to Named Entity
#     ie. < becomes &lt;, > becomes &gt; etc. This is required for <InputXML>
# Web Service XML is case sensitive, so be very exact
#
:tag=OutputResults
POST /CBMMugShotWebService/TokMugShot.asmx HTTP/1.1
Host: 100.15.5.70
Content-Type: text/xml; charset=utf-8
Content-Length: nnnnn
SOAPAction: "http://tempuri.org/CBMMugShotWebService/TokMugShot/GetMugshot"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
~ xmlns:xsd="http://www.w3.org/2001/XMLSchema"
~ xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<GetMugshot xmlns="http://tempuri.org/CBMMugShotWebService/TokMugShot">
<InputXML>
~%<?xml version="1.0" encoding="utf-8"?>
~%<InputParameters>
~% <add key="QueryString" value="${2}" />
~% <add key="ImageFormat" value="jpeg" />
~% <add key="TypeofImagesToBeDownloaded" value="10" />
~% <add key="RequiredImages" value="10" />
~% <add key="DisplayFields" value="2.015:,2.016:,2.017:,2.045:" />
~% <add key="subquery" value="" />
~% <add key="SortOrder" value="order by archivedate desc" />
~% <add key="NumberofRequestedResults" value="1" />
~% <add key="NumberofImagestobeDownloadedForEachType" value="1" />
~%</InputParameters>
~</InputXML>
<OutputXML></OutputXML>
</GetMugshot>
</soap:Body>
</soap:Envelope>
```

The Web Service template file is read and processed. Items like **${2}** would be replaced by the 2nd parameter of WEB-PARAMS, **${1}** would be replaced by the 1st parameter, etc. The ***nnnnn*** following Content-Length will be replaced with the correct numeric value for the length of the message after all merging of application supplied data has been completed. Note the blank line before **<?xml** is required to separate the HTTP header from the body of the message.

The resulting POST command is sent off to the declared 'Host:' and the result is read back and stored into WEB-BUF. The application would then need to scan WEB-BUF and interpret the results.

A tutorial using TSP

The TIP Sample Program (TSP) can be found in `$TIPROOT/src/tip` directory.

An XML template file for use by GETWEBSERIVCE called XMLTSP.xml can be found in the `$TIPROOT/tipsoa` directory after TIP/ix has been installed.

The XMLTSP.xml file is as follows:

```
POST /TSPWebService/TSP.asmx HTTP/1.1
Host: popa:15080
Content-Type: text/xml; charset=utf-8
Content-Length: nnnnn
SOAPAction: "http://tempuri.org/TSPWebService/TSP"

<?xml version="1.0" encoding="utf-8"?>
<soap-Env:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap-Env:Body>
<MenuFunc>${1}</MenuFunc>
<MenuKey>${2}</MenuKey>
</soap-Env:Body>
</soap-Env:Envelope>
```

When this is sent to another TIP/ix system which also has the XML capable version of TSP TIP/ix takes the transaction code from the last word of `SOAPAction:`.

The ${1} and ${2} get replaced by parameters. The code in TSP to invoke this looks like:

```
        77  PRM-LEN      PIC 9(4) COMP-4 VALUE 20.
        77  NUM-PRMS     PIC 9(4) COMP-4 VALUE 2.
        77  WEB-BUF-LEN  PIC 9(8) COMP-4 VALUE 30000.
        77  WEB-BUF-RTN  PIC S9(8) COMP-4 VALUE -1.
        77  XML-MAP-NAME PIC X(8).
        01  WEB-TAG      PIC X(48).
        01  WEB-BUF      PIC X(30000).
        01  WEB-SERVICE  PIC X(64) VALUE "XMLTSP".
        01  WEB-PARAMS.
            05 WEB-PARAM1  PIC X(20) VALUE " ".
            05 WEB-PARAM2  PIC X(20) VALUE "  ".
…
            IF SOAP-REQUEST
                GO TO 1200-SOAP-ROUTINE.
…
       *
       ***  Invoke Web Service
       *
        1200-SOAP-ROUTINE.
            MOVE "Web Service called" TO ERROR-MESSAGE
            PERFORM 9100-SEND-ERROR-MSG
            MOVE "XMLTSP"  TO WEB-SERVICE.
            MOVE "SH"      TO WEB-PARAM1.
            MOVE MENU-KEY  TO WEB-PARAM2.
```

```
           CALL "GETWEBSERVICE" USING WEB-SERVICE,
                                NUM-PRMS,
                                PRM-LEN,
                                WEB-PARAMS,
                                WEB-BUF,
                                WEB-BUF-LEN,
                                WEB-BUF-RTN.
           IF WEB-BUF-RTN < 0
               MOVE "Problem calling Web Service" TO ERROR-MESSAGE
               GO TO 0100-DISPLAY-MENU
           END-IF
           CALL "FIRSTXMLTAG" USING WEB-TAG,
                                WEB-BUF,
                                WEB-BUF-RTN.
           IF WEB-TAG = "NUM"
               MOVE WEB-BUF-LEN TO WEB-BUF-RTN
               CALL "TIPXML"  USING "G",
                                FULL-NAME,
                                MCS-DATA,
                                WEB-BUF,
                                WEB-BUF-RTN
               IF WEB-BUF-RTN < 0
                   MOVE WEB-BUF TO ERROR-MESSAGE
                   GO TO 0100-DISPLAY-MENU
               END-IF
               MOVE FULL-NAME         TO MCS-NAME
               MOVE " "               TO MCS-FILLER
               MOVE FULL-SIZE         TO MCS-COUNT
               MOVE TSPFL-KEY         TO MCS-CUST-NUM
               MOVE "Returned from Web Service" TO ERROR-MESSAGE
               PERFORM 9200-SEND-OUTPUT-MSG
               PERFORM 9300-GET-INPUT-MSG
               GO TO 0100-DISPLAY-MENU
           END-IF
           IF WEB-TAG = "ERROR"
           OR WEB-TAG = "FAULTSTRING"
               MOVE WEB-BUF-LEN TO WEB-BUF-RTN
               CALL "TIPXML"  USING "S",
                                WEB-TAG,
                                ERROR-MESSAGE,
                                WEB-BUF,
                                WEB-BUF-RTN,
               GO TO 0100-DISPLAY-MENU
           END-IF
           GO TO 0200-READ-MENU.
```

The CALL GETWEBSERVICE located the `XMLTSP.xml` template and merges in the parameters and sends it over to the host and port identifies by the ‘Host:’ line in the template file. To send to a different host just edit this line.

The FIRSTXMLTAG call grabs the first XML tag name in the returned data. If this is NUM then a TSPFILE record has been returned, otherwise it is some error message.

The following code as added to the 9300-GET-INPUT-MSG routine so that the transaction does not process more than one input message. This is just for safety.

```
      *  WebService: only process 1 input message and then terminate
           IF (PIB-TIPWEBSERVICE OR PIB-TYPE-WEBSERVICE)
           AND MSGI-COUNT > 1
               EXIT PROGRAM.
```

The following code as added to detect that it is running as a transactional web service and bypass the initial TIPMSGO of the prompt screen.

```
       0100-DISPLAY-MENU.
           IF PIB-TIPWEBSERVICE OR PIB-TYPE-WEBSERVICE
               MOVE MENU-NAME                  TO MCS-NAME
               GO TO 0200-READ-MENU.
```

So when TSP gets control as a Web Service it immediately does a TIPMSGI call. The MCS screen format name has been created as a set of XML data mapping rules using the following commands:

```
cd $TIPROOT/include
tipcblxml -m 'TF$TSP1_' XM-TSP1X
tipcblxml -m 'TF$TSP2_' XM-TSP2X
```

For example XM-TSP1X looks like:

```
      * This COPY Book is used to Define the XML input to TSP
      * This is for input to MCS Format  TF$TSP1A
      *
           05  MENU-FUNCTION    PIC X(2) IDENTIFIED BY "MenuFunc".
              88  SHOW-REQUEST            VALUE "D " "DI"
                                           "S " "SH" "SO".
              88  UPDT-REQUEST            VALUE "U " "UP".
           05  MENU-KEY         PIC X(8) IDENTIFIED BY "MenuKey".
```

You can see that the XML tags used here match up with those used in the **`XMLTSP.xml`** template file.

So the initial TIPMSGI call reads the input data and places it into the MCS-DATA area. The command is “SH” (i.e. Show a record) so TSP goes to read the record, copies the fields to MCS-DATA and does a TIPMSGO and then terminates. The TIPMSGO looks for the XML data mapping rules and with that converts the MCS-DATA area back into XML and sends back the response.

Tutorial Message Exchange

This is an example of transactional mode using a valid TSPFILE key the request is:

```
POST /TSPWebService/TSP.asmx HTTP/1.1
Host: popa:15080
Content-Type: text/xml; charset=utf-8
Content-Length: 328
SOAPAction: "http://tempuri.org/TSPWebService/TSP"

<?xml version="1.0" encoding="utf-8"?>
<soap-Env:Envelope
      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
     xmlns:xsd=http://www.w3.org/2001/XMLSchema
     xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap-Env:Body>
<MenuFunc>SH</MenuFunc>
<MenuKey>DEL00001</MenuKey>
</soap-Env:Body>
</soap-Env:Envelope>
```

Reply is:

```
HTTP/1.1 200 OK
Content-type: text/xml; charset=utf-8
Content-Length: 653

<?xml version="1.0"?>
<soap:Envelope
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<Num>DEL00001</Num>
<Company>DELTA LUGGAGE REPAIRS</Company>
<Adrs1>1620 ARIZONA WAY</Adrs1>
<Adrs2>TORONTO</Adrs2>
<Adrs3>DATA IS FROM POPA</Adrs3>
<Postal>N0B 2H0</Postal>
<Phone>4169898509</Phone>
<PoNum>TIP/IX</PoNum>
<Attn>MR. PETER MACKAY</Attn>
<DpMgr>MR. PETER MACKAY</DpMgr>
<Machine>UNI-80/5</Machine>
<Memory>3MEG</Memory>
<Disk>8470</Disk>
<Tape>6250 BPI</Tape>
<Terminals>45</Terminals>
</soap:Body>
</soap:Envelope>
```

Using an invalid TSPFILE key the request is:

```
POST /TSPWebService/TSP.asmx HTTP/1.1
Host: popa:15080
Content-Type: text/xml; charset=utf-8
Content-Length: 328
SOAPAction: "http://tempuri.org/TSPWebService/TSP"

<?xml version="1.0" encoding="utf-8"?>
<soap-Env:Envelope
      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
     xmlns:xsd=http://www.w3.org/2001/XMLSchema
     xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap-Env:Body>
<MenuFunc>SH</MenuFunc>
<MenuKey>RBW00045</MenuKey>
</soap-Env:Body>
</soap-Env:Envelope>
```

The reply is:

```
HTTP/1.1 500 Server Error
Content-type: text/xml; charset=utf-8
Content-Length: 370

<?xml version="1.0"?>
<soap-Env:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap-Env:Body>
<soap-Env:Fault>
<faultstring>Record not found</faultstring>
<faultcode>MCS Error</faultcode>
</soap-Env:Fault>
</Soap-Env:Body>
</Soap-Env:Envelope>
```

Using an invalid transaction code of TSPBAD the request is:

```
POST /TSPWebService/TSP.asmx HTTP/1.1
Host: popa:15080
Content-Type: text/xml; charset=utf-8
Content-Length: 328
SOAPAction: "http://tempuri.org/TSPWebService/TSPBAD"

<?xml version="1.0" encoding="utf-8"?>
<soap-Env:Envelope
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd=http://www.w3.org/2001/XMLSchema
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap-Env:Body>
<MenuFunc>SH</MenuFunc>
<MenuKey>DEL00001</MenuKey>
</soap-Env:Body>
</soap-Env:Envelope>
```

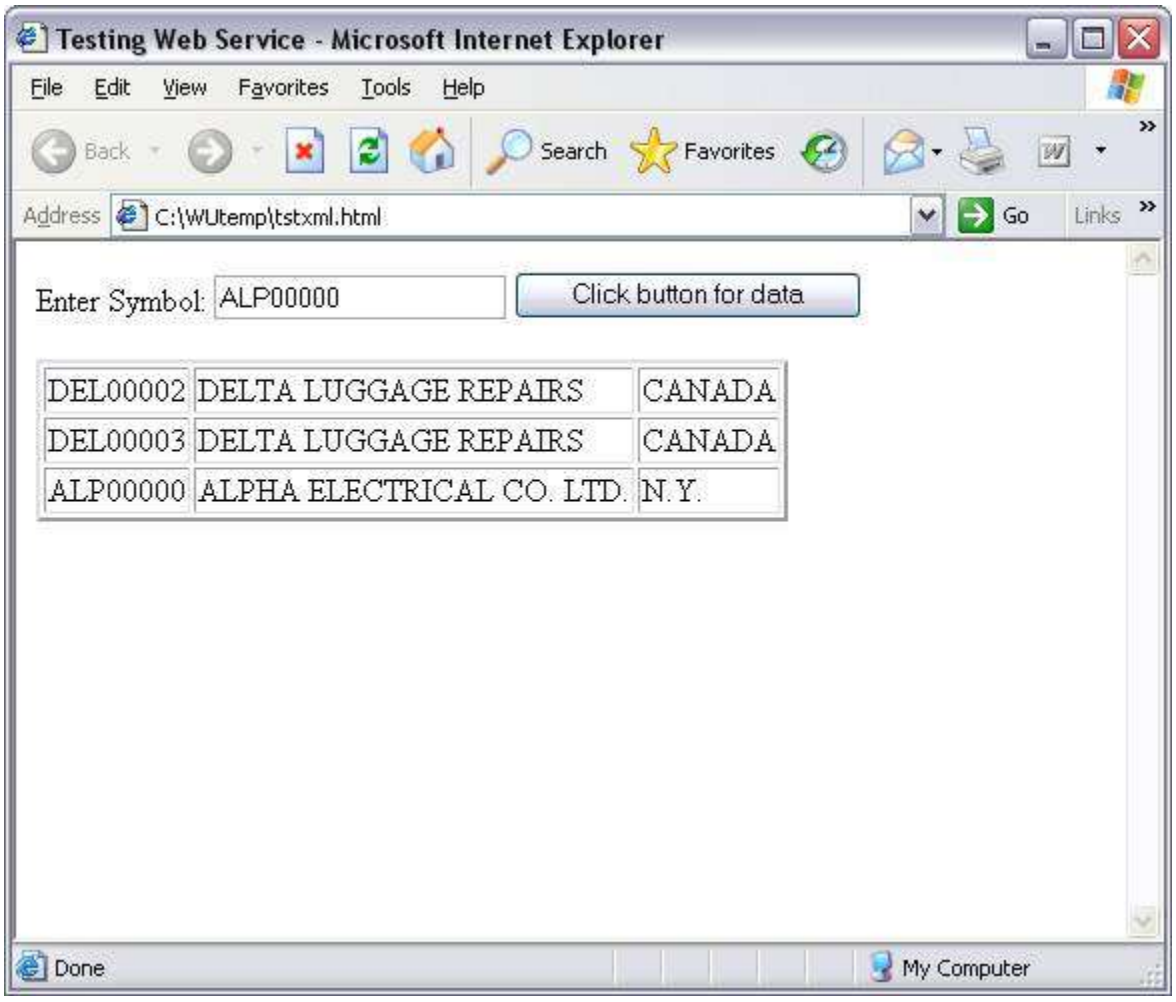
The reply is:

```
HTTP/1.1 500 Server Error
Content-type: text/xml; charset=utf-8
Content-Length: 396

<?xml version="1.0"?>
<soap-Env:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap-Env:Body>
<soap-Env:Fault>
<faultstring>Invalid transaction code! TSPBAD</faultstring>
<faultcode>Invalid Transaction</faultcode>
</soap-Env:Fault>
</Soap-Env:Body>
</Soap-Env:Envelope>
```

AJAX Invoking TIP/ix Web Service

Following is a very simple web page that is created using some JavaScript. Each time you key in a key for one of the TSPFILE records and click the button it will invoke TSP as a web service and then display some of the data when it comes back in a table.



The following is invoking TSP in transactional mode. The JavaScript for this example follows:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Testing Web Service</title>
<script>
var xmlhttp = null;
if (window.XMLHttpRequest) {
  xmlhttp = new XMLHttpRequest();
} else if(window.ActiveXObject) {
  xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}

function sendData()
 {
  xmlhttp.abort();
  var url ="http://192.168.1.22:8082/SOAPAction/TSP.tipx?MenuFunc=SH&MenuKey=" +
document.form1.TheKey.value;
  xmlhttp.open("GET",url,true);
  xmlhttp.onreadystatechange = getData;
  xmlhttp.send(null);
 }
function getData()
 {
 if((xmlhttp.readyState == 4) && (xmlhttp.status == 200))
 {
  var myXml = xmlhttp.responseXML;
  var xmlobject = null;
  if (window.ActiveXObject)
  {
    xmlobject = myXml.childNodes[1].firstChild;
  } else {
    xmlobject = myXml.childNodes[0].firstChild;
  }
  var table = document.getElementById("table1");
  var row = table.insertRow(table.rows.length);
  var cell1 = row.insertCell(row.cells.length);
  cell1.appendChild(getText("Num",xmlobject));
  var cell2 = row.insertCell(row.cells.length);
  cell2.appendChild(getText("Company",xmlobject));
  var cell3 = row.insertCell(row.cells.length);
  cell3.appendChild(getText("Adrs3",xmlobject));
  table.setAttribute("border", "2");
 }
}
function getText(tagName, xmlobject)
 {
  var tags = xmlobject.getElementsByTagName(tagName);
  var txtNode = null;
  if(window.ActiveXObject)
  {
    txtNode = document.createTextNode(tags[0].firstChild.text);
  } else {
    txtNode = document.createTextNode(tags[0].firstChild.textContent);
  }
 return txtNode;
 }
</script>
</head>
<body>
<form id="form1" name = "form1">
Enter Symbol: <input type="text" name="TheKey" id="TheKey" />
<input type="button" onclick="sendData()" value="Click button for data"/>
<br/> <br/>
<table id="table1">
</table>
</form>
</body>
</html>
```

C# Invoking TSP Web Service

The following is a very simple example of a C# program invoking the TSP program as a web service. Again this assumes, TIP/ix is configured to listen on port 8082. For you own system you would most likely have a different IP address or hostname and a different TCP/IP port. A screen shot of the program follows:

The sample C# code for the program follows:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Xml;

namespace TSP.Win
{
    /// <summary>
    /// TIP Sample Program form demonstrating how to connect
    /// to the TSP application via the TIPix TSP Web Service
    /// </summary>
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnGo_Click(object sender, EventArgs e)
        {
            // convert the alpha key to uppercase
```

```
            // and initialize controls showing the results
            txtKeyAlpha.Text = txtKeyAlpha.Text.ToUpper();
            txtCompany.Text = String.Empty;
            txtDpMgr.Text = String.Empty;

            XmlReader reader = null;
            try
            {
                // invoke the TIPix web service passing function and key inline
                string url =
String.Format("http://192.168.1.22:8082/SOAPAction/TSP.tipx?MenuFunc=SH&MenuKey
={0}", txtKeyAlpha.Text);
                reader = XmlReader.Create(url);
                // parse the xml and retrieve data
                ScanXmlData(reader);
            }
            catch (System.Exception x)
            {
                MessageBox.Show(x.Message);
            }
            finally
            {
                if (reader != null)
                    reader.Close();
            }
        }

        //
        // For each XML tag, this is called with the name & value
        //  mostly just copy the data to the form in the right spot
        //
        private void ProcessXmlData(string tagname, string tagvalue)
        {
            switch (tagname.ToLower())
            {
                case "errmsg":          // Could be from ERRMSG format
                    txtErrorMsg.Text = tagvalue;
                    break;
                case "faultstring":     // Could be a general Soap/Xml error
                    txtErrorMsg.Text = tagvalue;
                    break;
                case "company":
                    txtCompany.Text = tagvalue;
                    break;
                case "dpmgr":
                    txtDpMgr.Text = tagvalue;
                    break;
                default:
                    break;
            }
        }
```

```
        //
        // Scan entire XML document, For each tag/value call ProcessXmlData
        //
        private void ScanXmlData(XmlReader reader)
        {
            string tagName, tagValue;
            tagName = "Unknown";
            reader.MoveToContent();

            do
            {
                switch (reader.NodeType)
                {
                    case XmlNodeType.Element:
                        tagName = reader.Name;
                        break;

                    case XmlNodeType.Text:
                        // we have a complete node, process it
                        tagValue = reader.Value;
                        ProcessXmlData(tagName, tagValue);
                        break;
                }
                if (reader.EOF)
                    break;
            }
            while (reader.Read());
        }
    }
}
```

Soap Template files

The various template files used are stored in the $TIPROOT/tipsoa directory. The basic version is called default.reply and is used if there is nothing else found matching either the MCS screen name or the transaction code being used.

```
HTTP/1.1 200 OK
Content-type: text/xml; charset=utf-8
Cache-Control: no-cache
Content-Length: nnnnn

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
${reply}
</soap:Body>
</soap:Envelope>
```

When the above template is used the nnnnn following Content-Length gets replaced by that actual length of the data being sent and ${reply} is replaced by the MCS-DATA converted to XML.

Another example template file follows and let's say this file is called tsp.reply and the program running is the transaction code TSP.

```
HTTP/1.1 200 OK
Content-type: text/xml; charset=utf-8
Cache-Control: no-cache
Set-Cookie: ${sid}; Path=/
Content-Length: nnnnn

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLS
chema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<tran_id>${pib.trid}</tran_id>
<mcsname>${mcs.name}</mcsname>
${reply}
</soap:Body>
</soap:Envelope>
```

In the above example, ${pib.trid} would be replaced by the transaction code name (TSP) and then ${mcs.name} would be replaced by the MCS-NAME field used.

All meta-data names start with **${** and end with **}** and the possible names are:

Name	**Is replaced by**
sid	current Session-Id
pib.trid	PIB-TRID
pib.date	PIB-LONG-DATE
func	'msgo' if TIPMSGO was done 'msgov' if TIPMSGOV was done 'list' if TIPLIST was done
mcs.name	MCS-NAME
mcs.func	MCS-FUNCTION
mcs.fill	MCS-FILLER
mcs.count	MCS-COUNT
mcs.size	MCS-SIZE
mcs.error	'text' set via the TIPMSGEO call.
mcs.error:nn	nn bytes of the 'text' set via the TIPMSGEO call
reply	The MCS-DATA for MCS-COUNT bytes converted to XML (or JSON if there was Content-type: text/json)
reply.xml	The MCS-DATA for MCS-COUNT bytes converted to XML.
reply.json	The MCS-DATA for MCS-COUNT bytes converted to JSON

Client programming considerations

If operating in 'transactional mode' (which was the old way that TIP/ix worked) the client was only able to send to TIP/ix one HTTP SOAP message with XML data. The HTTP message could be either a GET or a POST format. TIP/ix would run the transaction program which could read the data using TIPMSGI, do some processing to prepare reply data and then send back one reply message using TIPMSGO. The reply message was an HTTP SOAP message with XML data inside the `<soap:body>`.

Now the default is 'conversational mode' so the old client side applications developed for 'transactional style' should continue to work however, the TIP/ix session will remain active and accept and process multiple HTTP SOAP exchanges without closing the TCP/IP socket. The COBOL program running under TIP/ix may need some code changes to check for PIB-TIPWEBSERVICECONV as well as PIB-TIPWEBSERVICE if the programs behavior depends on the terminal type.

But with 'conversational mode' the client may exchange many messages with the program running under TIP/ix. However the client must be careful to always save the most recent **Session_Id** value and return that on the next message sent to TIP/ix.

In addition the client must keep track of its conversation with the web service and know what data is expected to be sent next. If the COBOL program does some action based on a function key being pressed on a traditional terminal then the client of the web service may send in a function key value with the tag defined by `PARAM SOAPFUNCTIONTAG` in the `tipix.conf` file. The function key value is 'F#nn' where nn is two digits representing the function key. (F#00 indicates MSG WAIT key). His can be sent in either a POST or GET HTTP message. Example:

```
GET /?Function_Key=F%2300 HTTP/1.1
Accept: */*
Host: 192.168.1.32:8082
Connection: Keep-Alive
Cookie: MySession_Id=EsMFAQAAAEqf
```

Note the on a GET, %23 is actually the '#' character so this is sending 'F#00' to represent the MSG WAIT key.

TSP conversational Example

The following JavaScript code is written to interact with TSP running in conversational mode. The code saves the session-id it is sent and returns the value on the next message sent to TIP/ix for TSP. The test program also uses an SX command which tells TSP to display a record and then to a TIPLIST call. This is just done to demonstrate the interaction with TIPLIST.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Testing Web Service thru connection server</title>
<script>
var xmlhttp = null;
var sessid = null;
var islogin = null;
var logmsg = "User_Id=rjn&Password=r.norman&Computer_Name=RJNWIN7&";
var logxml = "
<User_Id>RJNX</User_Id><Password>xxxxxxx</Password><Computer_Name>RNXX7</Computer_Name>";
var hostadrs = "https://192.168.1.999:8085/";
if (window.XMLHttpRequest) {
  xmlhttp = new XMLHttpRequest();
} else if(window.ActiveXObject) {
  xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}

function pollData()
 {
  var url = hostadrs + "any_output";
  if(sessid != null)
         url = url + "?MySession_Id=" + sessid;
  xmlhttp.abort();
  xmlhttp.open("GET",url,true);
  xmlhttp.setRequestHeader("Accept", "text/xml");
  xmlhttp.onreadystatechange = getData;
  xmlhttp.send(null);
 }

function sendLogin()
 {
  var url = hostadrs + "services-tipix/TSP?" + logmsg ;
  islogin = 'true';
  logxml = "";
  logmsg = "";
  xmlhttp.abort();
  xmlhttp.open("GET",url,true);
  xmlhttp.setRequestHeader("Accept", "text/xml");
  xmlhttp.onreadystatechange = getData;
  xmlhttp.send(null);
 }

function sendLogoff()
 {
  var url = hostadrs + "services-tipix/TSP?" + logmsg + "Function_Key=F%2304";
  xmlhttp.abort();
  xmlhttp.open("GET",url,true);
  xmlhttp.onreadystatechange = getData;
  xmlhttp.send(null);
    elt = document.getElementById("Login");
    elt.value = "Press F5 to refresh browser";
    elt = document.getElementById("Logoff");
    elt.value = "Press F5 to refresh browser";
    elt = document.getElementById("ClickData");
```

```
    elt.value = "Press F5 to refresh browser";
 }

function sendDataGet()
 {
  xmlhttp.abort();
  var url = hostadrs + "services-tipix/TSP?" + logmsg +
"MenuFunc=SX&mcs_field=2&MenuKey=" + document.form1.TheKey.value;
  xmlhttp.open("GET",url,true);
  xmlhttp.onreadystatechange = getData;
  xmlhttp.send(null);
 }

function sendData()
 {
  var url = hostadrs + "services-tipix/TSP";
  var xmlsess = "";
  var nl = '\r\n';
  if(sessid != null)
        xmlsess = '<MySession_Id>' + sessid + '</MySession_id>';
  xmlhttp.abort();
  xmlhttp.open("POST",url,true);
  xmlhttp.setRequestHeader("Accept", "text/xml");
  xmlhttp.setRequestHeader("Content-Type", "text/xml");
  xmlhttp.setRequestHeader("SOAPAction", '"'+url+'"');
  xmlhttp.onreadystatechange = getData;
  xmlhttp.send(
        '<?xml version="1.0" encoding="UTF-8"?>' +
        '<soap:Envelope' +
        '  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"' +
        '  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"' +
        '  xmlns:xs="http://www.w3.org/2001/XMLSchema"' +
        '  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">' + nl +
        '   <soap:Body>' + nl +
        '       ' + xmlsess + logxml + nl +
        '      <MenuFunc>SX</MenuFunc>' +  nl +
        '      <MenuKey>' + document.form1.TheKey.value + '</MenuKey>' + nl +
        '   </soap:Body>' + nl +
        '</soap:Envelope>'
       );
  logxml = ' ';
 }

function sendListReply()
 {
  xmlhttp.abort();
  var url = hostadrs + "SOAPAction/TSP.tipx?" + logmsg +
"Mcs_Function=list&Tiplist_Data=Hello+World!";
  xmlhttp.open("GET",url,true);
  xmlhttp.onreadystatechange = getData;
  xmlhttp.send(null);
 }

function getData()
 {
 if((xmlhttp.readyState == 4) && (xmlhttp.status == 200))
 {
  var xmlobject = null;
  var elt = null;
  var dval = " ";
  var mcsName = "";
  var mcsFunc = "";
  sessid = getText("MySession_Id",xmlhttp.responseXML);
  if(sessid != null)
```

```
    logmsg = "MySession_Id=" + sessid + "&";
  if (window.ActiveXObject) {
    xmlobject = xmlhttp.responseXML.childNodes[1].firstChild;
  } else {
    xmlobject = xmlhttp.responseXML.childNodes[0].firstChild;
  }
  if(islogin == 'true') {
    islogin = 'done';
    elt = document.getElementById("Login");
    elt.value = "TSP is running";
    elt = document.getElementById("Logoff");
    elt.value = "Click to stop TSP and Logoff";
    console.log("Login completed");
    return;
  }
  if(xmlobject == null
  || !xmlobject.getElementsByTagName)
    xmlobject = xmlhttp.responseXML;
  mcsFunc = getText("Mcs_Function",xmlobject);
  console.log("mcsFunc is " + mcsFunc);
  if(mcsFunc == "msgo"
  || mcsFunc == "msgi") {
    mcsName = getText("Mcs_Name",xmlobject);
    console.log("mcsName is " + mcsName);
  }
  if(mcsName == 'TF$TSP1_') {
    elt = document.getElementById("ClickData");
    elt.value = "Ready! Enter key and Click";
    elt = document.getElementById("ListData");
    elt.value = "Do not Click this";
    return;
  }

  if(mcsFunc == "msgo") {
    var table = document.getElementById("table1");
    var row   = table.insertRow(table.rows.length);
    var cell1 = row.insertCell(row.cells.length);
    dval = getText("Num", xmlobject);
    cell1.innerHTML = dval;
    var cell2 = row.insertCell(row.cells.length);
    dval = getText("Company", xmlobject);
    cell2.innerHTML = dval;
    var cell3 = row.insertCell(row.cells.length);
    dval = getText("Adrs3", xmlobject);
    cell3.innerHTML = dval;
    table.setAttribute("border", "2");
    elt = document.getElementById("ClickData");
    elt.value = "Hold until after TIPLIST";
    elt = document.getElementById("ListData");
    elt.value = "Click now for TIPLIST";
  } else if(mcsFunc == "list") {
    var table = document.getElementById("table1");
    var row   = table.insertRow(table.rows.length);
    var cell1 = row.insertCell(row.cells.length);
    cell1.innerHTML = "<b>TipList</b>";
    var cell2 = row.insertCell(row.cells.length);
    cell2.innerHTML = "<b>Received</b>";
    var cell3 = row.insertCell(row.cells.length);
    cell3.innerHTML = "<b> ... </b>";
    table.setAttribute("border", "2");
    mcsFunc = " ";
    sendListReply();
    elt = document.getElementById("ClickData");
    elt.value = "Click now for next step";
```

```
  } else if(mcsFunc == "error") {
    var table = document.getElementById("table1");
    var row   = table.insertRow(table.rows.length);
    var cell1 = row.insertCell(row.cells.length);
    dval = getText("faultcode", xmlobject);
    cell1.innerHTML = dval;
    var cell2 = row.insertCell(row.cells.length);
    dval = getText("faultstring", xmlobject);
    cell2.innerHTML = dval;
    var cell3 = row.insertCell(row.cells.length);
    cell3.innerHTML = "<i>Error </i>";
    table.setAttribute("border", "2");
  } else if(mcsFunc == "msgi pending") {
    elt = document.getElementById("ClickData");
    elt.value = "Ready for key and Click";
  }
  } else if((xmlhttp.readyState == 4) && (xmlhttp.status == 500)) {
  var xmlobject = null;
  var dval = " ";
  var mcsFunc = getText("Mcs_Function",xmlhttp.responseXML);
  sessid = getText("MySession_Id",xmlhttp.responseXML);
  if(sessid != null)
    logmsg = "MySession_Id=" + sessid + "&";
  if (window.ActiveXObject) {
    xmlobject = xmlhttp.responseXML.childNodes[1].firstChild;
  } else {
    xmlobject = xmlhttp.responseXML.childNodes[0].firstChild;
  }

  if(xmlobject == null
  || !xmlobject.getElementsByTagName)
    xmlobject = xmlhttp.responseXML;
  if(mcsFunc == "error") {
    var table = document.getElementById("table1");
    var row   = table.insertRow(table.rows.length);
    var cell1 = row.insertCell(row.cells.length);
    dval = getText("faultcode", xmlobject);
    cell1.innerHTML = dval;
    var cell2 = row.insertCell(row.cells.length);
    dval = getText("faultstring", xmlobject);
    cell2.innerHTML = dval;
    var cell3 = row.insertCell(row.cells.length);
    cell3.innerHTML = "<b>500</b>";
    table.setAttribute("border", "2");
   }
 }
}

function getText(tagName, xmlobject)
 {
  var tags = xmlobject.getElementsByTagName(tagName);
  var txtNode = null;
  txtNode = xmlobject.getElementsByTagName(tagName)[0].firstChild.nodeValue;
  /*
  if(window.ActiveXObject) {
    txtNode = document.createTextNode(tags[0].firstChild.text);
  } else {
    txtNode = document.createTextNode(tags[0].firstChild.textContent);
  }
  */
  if(txtNode != null
  && txtNode.data) {
    return txtNode.data;
  }
```

```
  if(txtNode == null)
         return tagName + " not found";
 return txtNode;
 }
</script>

</head>

<body>
<form id="form1" name = "form1">
Enter Symbol: <input type="text" name="TheKey" id="TheKey" />
<input type="button" onclick="sendData()" id="ClickData" value="Click button for data"/>
<br/> Check for next message:
<input type="button" onclick="pollData()" id="ListData"  value="Check for TipList
message"/>
<br/>
<br/> Log in and start TSP :
<input type="button" onclick="sendLogin()" id="Login" value="Login and Start TSP"/>
<br/> Stop TSP & Logoff :
<input type="button" onclick="sendLogoff()" id="Logoff" value=" "/>
<br/>
<table id="table1">
</table>
</form>
</body>
</html>
```

The following is an example dialogue between the above JavaScript and TSP running on TIP/ix and the data is being transmitted thru the 'connection server' using HTTPS (i.e. SSL for data encryption.) The data is recorded by TIP/ix and the <INPUT> & <OUTPUT> makers denote the start of each message with <EOD> marking the end of the message text.

```
<INPUT> 000324 2015/01/03 10:48:47.87 192.168.1.998
GET /services-tipix/TSP?UserId=rjn&Pwrd=xxxx&Computer=RJNXX7& HTTP/1.1
Accept-Language: en-CA
Accept-Encoding: gzip, deflate
DNT: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: 192.168.1.999:8085
Client: 192.168.1.998
Connection: Keep-Alive
Accept: text/xml

<EOD>


<OUTPUT> 000663 2015/01/03 10:48:47.98 192.168.1.998 RJNXX7 SX01 TSP      TF$TSP0_
HTTP/1.1 200 OK
Content-type: text/xml; charset=utf-8
Cache-Control: no-cache
Set-Cookie: MySession_Id=A2sFAAAAAEon; Path=/
Content-Length: 510

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<termname>SX01</termname>
<Mcs_Function>msgo</Mcs_Function>
```

```
<Transaction_Id>TSP</Transaction_Id>
<Mcs_Name>TF$TSP0_</Mcs_Name>
<mcs_count>0</mcs_count>
<MySession_Id>A2sFAAAAAEon</MySession_Id>
<Error_Data>Test Error message for SOAP</Error_Data>
</soap:Body>
</soap:Envelope>

<EOD>


<INPUT> 000836 2015/01/03 10:48:57.49 RJNXX7 192.168.1.998 - SX01
POST /services-tipix/TSP HTTP/1.1
Content-Type: text/xml
Accept: text/xml
SOAPAction: "https://192.168.1.999:8085/services-tipix/TSP"
Accept-Language: en-CA
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: 192.168.1.999:8085
Client: 192.168.1.998
Content-Length: 434
DNT: 1
Connection: Keep-Alive
Cache-Control: no-cache

<?xml version="1.0" encoding="UTF-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <soap:Body>
      <MySession_Id>A2sFAAAAAEon</MySession_id>
      <MenuFunc>SX</MenuFunc>
      <MenuKey>DEL00000</MenuKey>
   </soap:Body>
</soap:Envelope>
<EOD>


<OUTPUT> 001063 2015/01/03 10:48:57.53 RJNXX7 192.168.1.998 SX01 TSP      TF$TSP2_
HTTP/1.1 200 OK
Content-type: text/xml; charset=utf-8
Cache-Control: no-cache
Set-Cookie: MySession_Id=A2sFAgAAAEol; Path=/
Content-Length: 910

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<termname>SX01</termname>
<Mcs_Function>msgo</Mcs_Function>
<Transaction_Id>TSP</Transaction_Id>
<Mcs_Name>TF$TSP2_</Mcs_Name>
<mcs_count>232</mcs_count>
<MySession_Id>A2sFAgAAAEol</MySession_Id>
<Error_Data>Test Error message for SOAP</Error_Data>
<Num>DEL00000</Num>
<Company>DELTA LUGGAGE REPAIRS</Company>
<Adrs1>1620 ARIZONA WAY</Adrs1>
<Adrs2>TORONTO</Adrs2>
<Adrs3>CANADA</Adrs3>
```

```
<Postal> </Postal>
<Phone>4169898509</Phone>
<PoNum>TIP/ix</PoNum>
<Attn>MR. PETER MACKAY</Attn>
<DpMgr>MR. PETER MACKAY</DpMgr>
<Machine>UNI-80/5</Machine>
<Memory>3MEG</Memory>
<Disk>8470</Disk>
<Tape>6250 BPI</Tape>
<Terminals>47</Terminals>
</soap:Body>
</soap:Envelope>

<EOD>


<INPUT> 000289 2015/01/03 10:48:59.51 RJNXX7 192.168.1.998 - SX01
GET /any_output?MySession_Id=A2sFAgAAAEol HTTP/1.1
Accept-Language: en-CA
Accept: text/xml
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: 192.168.1.999:8085
Client: 192.168.1.998
DNT: 1
Connection: Keep-Alive

<EOD>


<OUTPUT> 000889 2015/01/03 10:48:59.51 RJNXX7 192.168.1.998 SX01 TSP
HTTP/1.1 200 OK
Content-type: text/xml; charset=utf-8
Cache-Control: no-cache
Set-Cookie: MySession_Id=A2sFAwAAAEok; Path=/
Content-Length: 736

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<termname>SX01</termname>
<Mcs_Function>list</Mcs_Function>
<Transaction_Id>TSP</Transaction_Id>
<MySession_Id>A2sFAwAAAEok</MySession_Id>
<Tiplist_Data>
<item>Beer </item>
<item>Cardhu </item>
<item>Rum </item>
<item>Imported wine </item>
<item>Old sailor </item>
<item>eXport beer </item>
<item>iMported beer </item>
<item>Cognac VSOP </item>
<item>Scotch </item>
<item>Single malt </item>
<item>12 year old </item>
<item>How old am i? </item>
</Tiplist_Data>
</soap:Body>
</soap:Envelope>

<EOD>
```

```
<INPUT> 000337 2015/01/03 10:48:59.52 RJNXX7 192.168.1.998 - SX01
GET
/SOAPAction/TSP.tipx?MySession_Id=A2sFAwAAAEok&Mcs_Function=list&Tiplist_Data=Hello+World
! HTTP/1.1
Accept: */*
Accept-Language: en-CA
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: 192.168.1.999:8085
Client: 192.168.1.998
DNT: 1
Connection: Keep-Alive

<EOD>


<OUTPUT> 000663 2015/01/03 10:48:59.52 RJNXX7 192.168.1.998 SX01 TSP      TF$TSP0_
HTTP/1.1 200 OK
Content-type: text/xml; charset=utf-8
Cache-Control: no-cache
Set-Cookie: MySession_Id=A2sFBAAAAEoj; Path=/
Content-Length: 510

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<termname>SX01</termname>
<Mcs_Function>msgo</Mcs_Function>
<Transaction_Id>TSP</Transaction_Id>
<Mcs_Name>TF$TSP0_</Mcs_Name>
<mcs_count>0</mcs_count>
<MySession_Id>A2sFBAAAAEoj</MySession_Id>
<Error_Data>Test Error message for SOAP</Error_Data>
</soap:Body>
</soap:Envelope>

<EOD>


<INPUT> 000837 2015/01/03 10:49:09.84 RJNXX7 192.168.1.998 - SX01
POST /services-tipix/TSP HTTP/1.1
Content-Type: text/xml
Accept: text/xml
SOAPAction: "https://192.168.1.999:8085/services-tipix/TSP"
Accept-Language: en-CA
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: 192.168.1.999:8085
Client: 192.168.1.998
Content-Length: 435
DNT: 1
Connection: Keep-Alive
Cache-Control: no-cache

<?xml version="1.0" encoding="UTF-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <soap:Body>
```

```
      <MySession_Id>A2sFBAAAAEoj</MySession_id>
      <MenuFunc>SX</MenuFunc>
      <MenuKey>DEL00001</MenuKey>
   </soap:Body>
</soap:Envelope>
<EOD>


<OUTPUT> 001063 2015/01/03 10:49:09.84 RJNXX7 192.168.1.998 SX01 TSP      TF$TSP2_
HTTP/1.1 200 OK
Content-type: text/xml; charset=utf-8
Cache-Control: no-cache
Set-Cookie: MySession_Id=A2sFBQAAAEoi; Path=/
Content-Length: 910

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<termname>SX01</termname>
<Mcs_Function>msgo</Mcs_Function>
<Transaction_Id>TSP</Transaction_Id>
<Mcs_Name>TF$TSP2_</Mcs_Name>
<mcs_count>232</mcs_count>
<MySession_Id>A2sFBQAAAEoi</MySession_Id>
<Error_Data>Test Error message for SOAP</Error_Data>
<Num>DEL00001</Num>
<Company>DELTA LUGGAGE REPAIRS</Company>
<Adrs1>1620 ARIZONA WAY</Adrs1>
<Adrs2>TORONTO</Adrs2>
<Adrs3>CANADA</Adrs3>
<Postal> </Postal>
<Phone>4169898509</Phone>
<PoNum>TIP/ix</PoNum>
<Attn>MR. PETER MACKAY</Attn>
<DpMgr>MR. PETER MACKAY</DpMgr>
<Machine>UNI-80/5</Machine>
<Memory>3MEG</Memory>
<Disk>8470</Disk>
<Tape>6250 BPI</Tape>
<Terminals>47</Terminals>
</soap:Body>
</soap:Envelope>

<EOD>


<INPUT> 000289 2015/01/03 10:49:11.54 RJNXX7 192.168.1.998 - SX01
GET /any_output?MySession_Id=A2sFBQAAAEoi HTTP/1.1
Accept-Language: en-CA
Accept: text/xml
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: 192.168.1.999:8085
Client: 192.168.1.998
DNT: 1
Connection: Keep-Alive

<EOD>


<OUTPUT> 000889 2015/01/03 10:49:11.55 RJNXX7 192.168.1.998 SX01 TSP
HTTP/1.1 200 OK
```

```
Content-type: text/xml; charset=utf-8
Cache-Control: no-cache
Set-Cookie: MySession_Id=A2sFBgAAAEoh; Path=/
Content-Length: 736

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<termname>SX01</termname>
<Mcs_Function>list</Mcs_Function>
<Transaction_Id>TSP</Transaction_Id>
<MySession_Id>A2sFBgAAAEoh</MySession_Id>
<Tiplist_Data>
<item>Beer </item>
<item>Cardhu </item>
<item>Rum </item>
<item>Imported wine </item>
<item>Old sailor </item>
<item>eXport beer </item>
<item>iMported beer </item>
<item>Cognac VSOP </item>
<item>Scotch </item>
<item>Single malt </item>
<item>12 year old </item>
<item>How old am i? </item>
</Tiplist_Data>
</soap:Body>
</soap:Envelope>

<EOD>


<INPUT> 000337 2015/01/03 10:49:11.58 RJNXX7 192.168.1.998 - SX01
GET
/SOAPAction/TSP.tipx?MySession_Id=A2sFBgAAAEoh&Mcs_Function=list&Tiplist_Data=Hello+World
! HTTP/1.1
Accept: */*
Accept-Language: en-CA
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: 192.168.1.999:8085
Client: 192.168.1.998
DNT: 1
Connection: Keep-Alive

<EOD>


<OUTPUT> 000663 2015/01/03 10:49:11.58 RJNXX7 192.168.1.998 SX01 TSP      TF$TSP0_
HTTP/1.1 200 OK
Content-type: text/xml; charset=utf-8
Cache-Control: no-cache
Set-Cookie: MySession_Id=A2sFBwAAAEog; Path=/
Content-Length: 510

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<termname>SX01</termname>
<Mcs_Function>msgo</Mcs_Function>
```

```
<Transaction_Id>TSP</Transaction_Id>
<Mcs_Name>TF$TSP0_</Mcs_Name>
<mcs_count>0</mcs_count>
<MySession_Id>A2sFBwAAAEog</MySession_Id>
<Error_Data>Test Error message for SOAP</Error_Data>
</soap:Body>
</soap:Envelope>

<EOD>


<INPUT> 000312 2015/01/03 10:49:14.69 RJNXX7 192.168.1.998 - SX01
GET /services-tipix/TSP?MySession_Id=A2sFBwAAAEog&Function_Key=F%2304 HTTP/1.1
Accept: */*
Accept-Language: en-CA
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: 192.168.1.999:8085
Client: 192.168.1.998
DNT: 1
Connection: Keep-Alive

<EOD>


<OUTPUT> 000553 2015/01/03 10:49:14.69 RJNXX7 192.168.1.998 - SX01
HTTP/1.1 200 OK
Content-type: text/xml; charset=utf-8
Cache-Control: no-cache
Set-Cookie: MySession_Id=A2sFCAAAAEov; Path=/
Content-Length: 400

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<termname>SX01</termname>
<Mcs_Function>erase</Mcs_Function>
<Transaction_Id>TSP</Transaction_Id>
<MySession_Id>A2sFCAAAAEov</MySession_Id>
</soap:Body>
</soap:Envelope>

<EOD>
```