MGS. Inc.

Computer Business Solutions

# MGS Web Services for Unisys MCP Systems (MGSWeb)

# Reference Manual

## Version 3.12

September, 2016 - Rev 3

**MGS Web Services for Unisys MCP Systems Reference Manual**

Version 3.12



## Disclaimer

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THIS DOCUMENT.

This document describes the current release of the MGS Web Services for Unisys MCP Systems. The MGS Web Services for Unisys MCP Systems Software is a proprietary product suite offered for licensing by MGS, Inc. Use of this software requires that a license agreement be signed with MGS, Inc or one of its distributors. No part of this document may be copied, distributed, or transmitted in any form or by any means, mechanical or electronic, without the express written permission of MGS, Inc.

MGS, Inc. believes the information presented in this document is accurate and reliable. However, MGS, Inc. assumes no responsibility for any consequences arising from the use of the MGSWeb software, this document or the information contained herein. MGS, Inc. reserves the right to revise the contents of this publication without obligation to notify any person of such revisions.

All questions and comments concerning this document should be directed to:

## MGS, Inc. Customer Support

MGS, Inc.
583 Southlake Boulevard
Suite A
Richmond, VA 23236

Phone: 804-379-0230
Fax: 804-379-1299
E-mail: support@mgsinc.com

September, 2016 - Rev 3

# 1. Overview

**The MGS Web Services for Unisys MCP software** is a general purpose Web Services implementation that runs under MCP control rather than outboard in a middle tier computer. It provides an MCP system with the native ability to both process inbound Web Services requests by application programs on other computers (WS-In) as well as giving MCP applications the ability to make outbound Web Services requests of other computers (WS-Out). An additional facility of MGS Web Services is that it provides MCP applications with the ability to also make outbound HTTP protocol requests to acquire information from other computers (HTTP-Out).

For WS-In, Web Services use eXensible Markup Language (XML) and the Simple Object Access Protocol (SOAP) to encode requests and responses, and use the HyperText Transport Protocol (HTTP) to deliver the request/response data to a network connected Web Services client application. WS-In Web Services on the Unisys MCP system can use either Unisys' Web Transaction Server (ATLAS) or MGS' Web Services server as its HTTP server.

For WS-Out and HTTP-Out, MGS Web Services provides its own outbound HTTP client software to acquire request/response data from a network connected Web Server or Web Services server.

The transaction volume supported by MGS Web Services is limited only by the capacity level of the MCP platform and the time required to establish TCP connections. For higher transaction volumes, the MGS Web Services software supports "persistent" connections, so multiple transactions can be done over the same socket connection.

MGSWeb also supports the ability to call both inbound and outbound web services over Secure Socket Layer (SSL) connections. Inbound SSL web services are provided by routing the inbound web services call through the Unisys Web Transaction Server (ATLAS). Outbound web services connections using SSL are supported natively by the MCP-based MGS Web Services software.

## 1.1 WS-In Inbound Web Services

The WS-In inbound Web Services software allows a MCP system to provide other network connected computer systems direct access to COMS transactions and DMSII databases using Web Services, without requiring a middle tier system.

Figure 1 below shows the basic operation of WS-In.

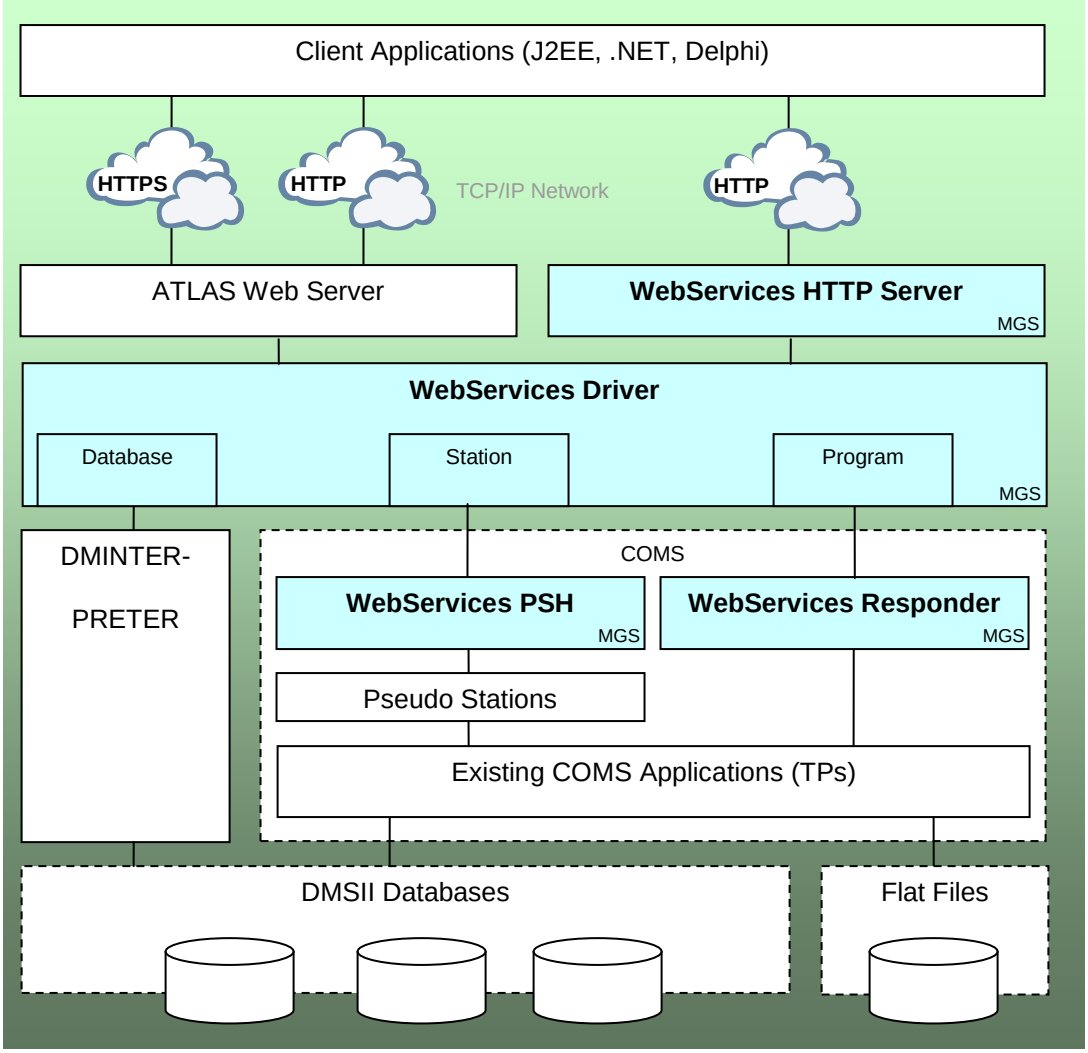



***Figure 1 – MGSWeb WS-In Data Flow***

WS-In Web Services for Unisys MCP currently supports three types of transactions: *Station*, *Program*, and *Database*.

- **Station** transactions use a COMS Protocol Specific Handler (PSH) station to submit the transaction to a COMS application.

  The incoming Web Services requests are reformatted from XML (so they appear to have come from a station) and transmitted to COMS via a PSH station. The PSH station responses are then "screen-scraped" and converted into XML web service responses.

  A Web Services client application can allocate a specific PSH station for its exclusive use for multiple transactions, or it can use any available station for each transaction (for stateless transactions).

Most COMS applications can be accessed via Station transactions without any modifications.

- **Program** transactions use a special COMS program (included with the Web Services software) to do TP-to-TP messages.

  The incoming requests are converted from XML into flat records with fixed-size fields (COBOL "01"s) and sent to a COMS application. The application's response is then converted into an XML web service response.

  The COMS application must be specially coded to handle the TP-to-TP messages, but this method is more efficient than the Station transaction path.

- **Database** transactions use the DMSII interpretive interface to retrieve records from a DMSII database.

  The incoming requests are converted from XML to an interpretive interface "condition", used to execute a FIND against the database. The resulting record is then converted into an XML Web Services response.

The conversion of the incoming requests and the generation of the responses are all controlled by *host configuration* files, which are generated by the Web Services Utility (WSUtility) on a Windows PC and transferred to the host.

The WSUtility program also generates Web Services Definition Language (WSDL) files that describe the COMS Web Services. The WSDL files can be imported by programming environments (such as Visual Studio .NET or Delphi), so access to the web service transactions is very simple.

## 1.2 WS-Out and HTTP-Out Outbound Web Services

WS-Out and HTTP-Out Web Services for Unisys MCP supports two types of outbound service calls: WSDL-based Web Services (which are called a "Web Service," or WS-Out), and non-WSDL-based Web Services (which are called an "HTTP Service," or HTTP-Out).

For WS-Out, the WSUtility program is used to read the desired WSDL and to generate the necessary host configuration as well as generating COPY libraries that can be included into the calling application program. For HTTP-Out, where there is no WSDL file, the WSUtility is used to define both the HTTP and the application data formats before generating the host configuration and application COPY library files.

Figure 2 below shows the basic operation of WS-Out and HTTP-Out.

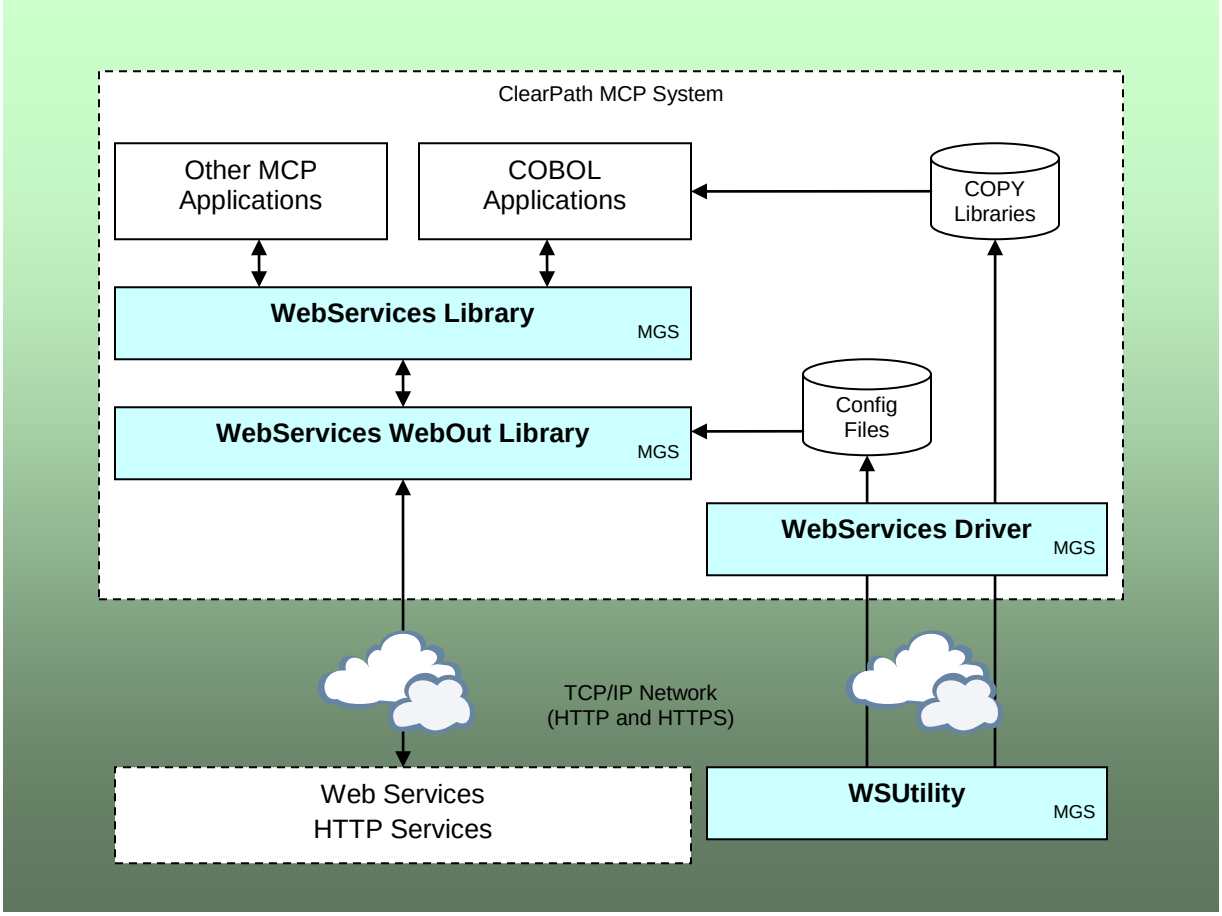



***Figure 2 – MGSWeb WS-Out and HTTP-Out Data Flow***

## 1.3 MGSWeb Authentication Extensions

For the WS-Out and HTTP-Out features the MGSWeb software supports the following authentication protocols:

- No authentication
- HTTP Basic
- HTTP Digest
- Windows NTLM

For authentication, the application program must supply the username and password to be used for the WS-Out or HTTP-Out call. The MGSWeb software will automatically perform the authentication required by the server.

# 2. Software Installation

MGSWeb software can be delivered in two ways – on a PC format CD or, for electronic distribution, in a Windows Compressed folder (ZIP file). There are two main steps to Web Services installation –installing the WSUtility program on a Windows PC and installing the Web Services code files on the host. Both the folder containing the PC software and the wrapped file containing the host software are on the Web Services release CD or in electronic distribution ZIP file.

## 2.1 Documentation

Complete documentation for MGS Web Services (this manual) can be found in the file `MGS MCP Web Services Reference Manual.pdf`, located in the root directory of the release CD or electronic release ZIP file.

## 2.2 Windows PC Installation

The Web Services for Unisys MCP release CD contains the Web Services Utility (WSUtility), which is a Windows program requiring Windows XP or newer. WSUtility performs the following functions:

- Generates host configuration files
- Generates WSDL files
- Optionally, generates ASP projects for Station transaction projects
- Allows you to modify project settings
- Allows you to test WS-In transactions
- Allows you enable or disable project-specific host tracing

WSUtility is used for all types of WS-In transactions, as well as for WS-Out and HTTP-Out transactions.

To install WSUtility, **copy the `WSUTILITY` folder from the MGSWeb release CD or ZIP file to your hard drive.** You may also want to add a shortcut to your desktop or Start menu that references the `WSUtility.exe` file contained in that folder, so the program can be run easily (make sure that the working directory of the shortcut is the directory in which the `WSUtility.exe` program is located).

| | |
|---|---|
| ***Note***: | For a WS-In Station Transaction Web Service, the Unisys Screen Object Modeling Studio (SOMS) software is needed for the first step where the screen capture is done. Hence, you will need to install the SOMS software before attempting to create a Station Transaction Web Service. This can be found on the Screen Object Modeling Studio CD that comes as part of your Unisys MCP Release CDs. **Simply place the SOMS CD in your CD drive and follow the Installer Wizard instructions**. |

The release media contains the following files in the `WSUTILITY` folder:

**`WSUtility.exe`**
: This is the Web Services Utility executable file.

**`webservices.css`**
: This is a copy of the Web Services Cascading Style Sheet (CSS) file from the AspFiles directory; it allows the host screen images to be displayed on WSUtility's Screen Image page correctly.

**`logo.gif`**
: This is the MGS, Inc graphic displayed on the opening WSUtility screen.

**`AspFiles\*.*`**
: This directory on the release media contains the skeleton files for the ASP project that WSUtility can generate.

**`Samples\*.*`**
: This directory on the release media contains the sample programs and supporting files. See Section 11 for details on the sample programs.

## 2.3 Host Installation

The installation of Web Services on the Unisys MCP host consists of uploading the wrapped file `MGSWEB.CON` (host code files container) to the host, unwrapping the files, optionally configuring either ATLAS or the MGS Web Server, and, if you are not using ATLAS, starting the Web Services WFL.

The wrap container `MGSWEB.CON` can be found in the root directory of the release CD or the electronic distribution ZIP file. You must use a binary file transfer to get the wrap container to the host; the easiest way to do this is using Windows Explorer to drag and drop the file to a Unisys MCP share.

To unwrap the code files, use the following command from a CANDE session logged on to a privileged usercode:

```
WFL UNWRAP *= AS (<usercode>)= OUTOF "MGSWEB.CON"
      TO <family> (PACK, RESTRICTED=FALSE)
```

where <usercode> is the usercode you want to install Web Services to and <family> is the family you want the Web Services files to install on.

**Configuring the Web Server Port Number**

If you are going to use the MGS Web Server, the default port that the web server listens on is 8080; if you wish to change this, you must edit the WEBSERVICES/SERVER/PARAM file (this file will be called WEBSERVICES/SERVER/NEW/PARAM for new installations). To do this, log on to the usercode you installed the Web Services code files to, get the WEBSERVICES/SERVER/PARAM file, change the "port" line in the file from 8080 to the port number you wish to use, and save the file.

**Release File List**

The following host files are included in the release:

**WEBSERVICES/DRIVER**

The main inbound Web Services code, including the XML parser and the SOAP processor.

**WEBSERVICES/HOSTCONFIG/=**

This directory contains host configuration files, generated by WSUtility, for all defined Web Services. The release includes host configuration files for the sample programs. See Section 11 for details on the sample programs.

**WEBSERVICES/PSH**

A COMS Protocol Specific Handler (PSH) that provides DRIVER with access to COMS stations.

**WEBSERVICES/SERVER**

SERVER is a general-purpose TCP/IP server, referred to in this document as the "Web Services Server".

**WEBSERVICES/SERVER/NEW/PARAM**

The default parameter file for the SERVER program; it contains the port number that the SERVER will listen on (port number 8080 by default). This file will be renamed WEBSERVICES/SERVER/PARAM the first time the Web Services software is brought up.

**WEBSERVICES/HTTP/SERVER**

The HTTP-specific server library called by the SERVER program.

**WEBSERVICES/INTERFACE**

This code file is the interface library between the DRIVER and RESPONDER programs; it also manages the MGSWeb log file and host tracing for all modules.

**WEBSERVICES/LIBRARY**

This library is used by client MCP programs to call outbound Web Services using WS-Out.

**WEBSERVICES/WEBOUT**

This library implements the outbound WS-Out and HTTP-Out functionality called by `WEBSERVICES/LIBRARY`.

**WEBSERVICES/RESPONDER**

RESPONDER is a COMS program that sends and receives TP-to-TP COMS messages for Web Services Program transactions.

**WEBSERVICES/SAMPLES/=**

This directory contains the WS-In and WS-Out sample files. See Section 11 for details.

**WEBSERVICES/USERFILES/=**

This directory contains COPY library files, generated by WSUtility, for both WS-In Program web services and WS-Out and HTTP-Out web services. The release includes COPY libraries for the sample WS-Out program. See Section 11 for details on the sample programs.

**WEBSERVICES/WFL**

This WFL file starts the Web Services Server (SERVER).

## 2.4 Host License Key

To use the MGSWeb software you must acquire an AccessKey from MGS, Inc. The AccessKey is then specified to the Host software using the ACCESSKEY command in the `WEBSERVICES/PARAM` file. See Section 3.3 for details. The AccessKey is used to enable/disable the following product features:

- Incoming Web Services (WS-In)
- Outgoing Web Services (WS-Out)
- Outgoing HTTP Services (HTTP-Out)
- Authentication Extensions (Auth)

The AccessKey related messages that may be displayed are as follows:

*This software will expire in <number> days*

When running in trial mode (no AccessKey), or if your AccessKey is set to expire in less than 36 days, this message will be logged once a day. Web Services will still be available.

*This software has expired*

When you have either exhausted your trial period or your AccessKey has expired, this message will be logged every time you attempt to initiate the MGSWeb software. Web Services will not be available.

*The specified key is not valid for this system*

If your AccessKey does not match either your system serial number or system type this message will be logged every time you attempt to initiate the MGSWeb software. Web Services will not be available.

Contact MGS Support immediately if you are having any problems with your AccessKey.

## 2.5 SSL Considerations

MGSWeb supports the ability to call both inbound and outbound web services over Secure Socket Layer (SSL) connections. Inbound SSL web services are provided by routing the inbound web services call through the Unisys Web Transaction Server (ATLAS). Outbound SSL for WS-Out and HTTP-Out transactions is supported natively by the MCP based MGSWeb software.

You can configure the Unisys Web Transaction Server (ATLAS) to provide SSL connections for inbound web services connections, and you can use the Web Services Utility (WSUtility) to configure SSL for outbound connections.

Additionally, WSUtility supports both SSL and non-SSL connections to the MCP system to transfer host configuration and COBOL include files to the host, and to test inbound web services transactions. WSUtility also allows you to specify a client certificate it can use when connecting to the MCP system if ATLAS is configured to use two-way SSL.

MGSWeb SSL requirements:

- MGSWeb release 2.00 or later
- Unisys ClearPath Secure Transport software installed and configured on the MCP system
- For outbound: NPO versions 51.151 or 52.125 installed on the MCP system
- For inbound: an ATLAS defined SSL port for incoming MGSWeb transactions
- Required certificates or certificate authorities installed on the MCP system

### 2.5.1 Inbound SSL Transactions

MGSWeb supports SSL connections for inbound transactions via the ATLAS web server (the MGS Web Server included with MGSWeb does not support SSL). You must have installed and configured the Unisys ClearPath Secure Transport software and the necessary server certificates on your MCP system before you can use SSL. Please see the Unisys documentation for details on setting up SSL on your host system.

Next, to have ATLAS provide SSL connections for MGSWeb web services transactions, you must use Unisys' Site Manager to add an SSL port to the web site where you have defined the MGSWeb virtual directory, or add another MGSWeb virtual directory to a web site that already has an SSL port. For details on processing MGS web services through ATLAS, see section 3.2 of this document.

Once you have made the ATLAS configuration changes and updated ATLAS' configuration files on the host, MGSWeb transactions will then be accessible via secure sockets. You will have to update your Station, Program and Database web service project files using WSUtility to specify the secure port number and check the SSL option (and change the virtual directory, if necessary), and then re-generate the WSDL files and the host configuration files for each project.

WSUtility supports SSL connections to the host for transferring MGSWeb host configuration and include files, and for testing. WSUtility also supports specifying a client certificate when connecting to the MCP system if ATLAS is configured to require two-way SSL.

### 2.5.2 Outbound SSL Transactions

MGSWeb host software supports SSL connections outbound from the host to the server providing a web service or HTTP service. WSUtility also supports SSL connections to the host for transferring MGSWeb host configuration files and COBOL include files.

***Note***: For the MGSWeb outbound SSL to work, you must have installed and configured the Unisys ClearPath Secure Transport software on your MCP system.

***Note***: For proper SSL operation you must first install a fix for the MCP networking software. The required fix is in NPO version 51.151 (PRI 9777946) and in NPO version 52.125 (PRI 9777423).

To enable SSL for outbound connections, use WSUtility to set the secure port number and the SSL option for each Web Service and HTTP Service project file, and then re-generate the host configuration and COBOL include files. You may also have to reload the WSDL for Web Services, or change the Server URL for HTTP Services, for any existing service that was using a non-secure port that has been changed to a secure port.

By default, the MCP system does not come with any Certificate Authorities (CAs) pre-configured. Hence, you will have to use Unisys' Security Center software to ensure that the "ROOT" trusted store under the *MCP Cryptographic Services Manager* contains the issuer's certificate for any SSL sites that you want to access via outbound MGSWeb transactions. Please see the Security Center's help file for further information on installing certificates to the ROOT trusted store.

If the web server that provides the web service you are calling requires a client certificate (that is, the web server uses two-way SSL), your application can also specify the name of a client certificate "key container" when calling Web Services. The key must have already been created in Security Center's "Socket Keys" container before it can use used by Web Services.

# 3. Host Configuration

Web Services for Unisys MCP can use either the included Web Services server, or Unisys' Web Transaction Server (ATLAS) as its HTTP server.

Both the MGS Web Services server and ALTAS support high-volume "persistent" connections. ATLAS also supports persistent connection "pipelining," as well as Secure Socket Layer (SSL) connections for WS-In inbound transactions. The current version of the MGS Web Services server does not support pipelining or SSL.

Depending on your desired configuration, you may have to configure one or more of the following programs:

- The Web Services server, or Unisys's ATLAS server.
- The Web Services DRIVER program.
- The Web Services RESPONDER program.

The following sections detail the configuration of the HTTP servers, and the DRIVER and RESPONDER programs.

## 3.1 Configuring the Web Services Server

The `WEBSERVICES/SERVER/PARAM` file included with the release files uses 8080 as the Web Services port, and allows up to twenty concurrent connections. If you want to change these defaults, use CANDE to modify this file. (If you have not yet run the Web Services host software, you'll have to modify the `WEBSERVICES/SERVER/NEW/PARAM` file.)

The parameter file contains the following statements:

```
service WEBSERVICES
  port 8080,
  type STREAM,
  library "WEBSERVICES/HTTP/SERVER",
  dontwait,
  translate,
  maxsubfiles 20;
```

The port number is set using the **port** statement; you can change it to any number between 1 and 65535.

The maximum number of concurrent connections is set with the **maxsubfiles** statement; you can change this to any number between 1 and 256.

You should not change any of the other statements in the file.

***Note:*** The commas between statements and the terminating semi-colon are required.

After making changes to the parameter file, you must terminate the SERVER program (using the <*mix number*> *AX STOP* command) and restart it for the changes to take effect.

## 3.2 Configuring Unisys' Web Transaction Server (ATLAS)

If you use the ATLAS server instead of the Web Services server, you must configure ALTAS with an application and a virtual directory for Web Services. By default, the virtual directory is assumed to be **HostWebServices**.

Follow these steps to add the Web Services application and virtual directory to ATLAS' Site Manager:

1. Access the ATLAS Admin interface (port 2488) via a web browser; you will have to provide a valid MCP usercode and password that is allowed to access the Admin interface.

2. Start the Site Manager by clicking on the Site Manager link; note that Site Manager requires the Sun Java Virtual Machine (JVM).

3. Log on to the provider you want to add the Web Services application and virtual directory to; ordinarily, you would add them to the ATLASSUPPORT provider.

4. Click the server name in the left panel.

5. On the *Applications* tab on the right panel, click the **Add** button to add a new application.

6. Enter the following values on the Application dialog:

   | | |
   |---|---|
   | *Application Name:* | HOSTWEBSERVICES |
   | *API Type:* | AAPI |
   | *Authentication Method:* | None |
   | *Access Type:* | Title |

   Enter the name of the DRIVER program in "path" format in the *Title* edit box. If you copied the Web Services files un-usercoded to DISK, the title would be "`/webservices/driver`". If you copied the files under usercode (WS) on PACK, the title would be "`/-/pack/usercode/ws/webservices/driver`".

7. Click the **OK** button.

8. Next, click the web site you want to add the virtual directory to (usually "Default Web Site") in the left panel; the *Web Site* page should then be displayed.

9. Click the **New** button under the left panel, and enter the name of the virtual directory ("/hostwebservices/" is recommended) in the *New Virtual Directory* dialog.

10. The *Virtual Directory* page is then displayed. Enter the following settings on this page:

    *Directory Type:* An application
    *Application Name:* The application you added above (HOSTWEBSERVICES)

11. Click **Close**, and **Yes** to the "save changes?" prompt, and **OK** to the "restart the server?" prompt.

The ATLAS server will be restarted, and the new application and virtual directory will then be available. ATLAS will automatically start the DRIVER program when a request is received for the new virtual directory.

If you add the new MGSWeb virtual directory to a web site that has an "SSL Port" defined (step 5 above), applications will be able to do WS-In transactions over an SSL connection.

***Note:*** In order for inbound SSL connections to work, you must have installed and configured the *Unisys ClearPath Secure Transport* software on your MCP system. Please see the Unisys documentation for details on setting up Secure Transport on your host system.

## 3.3 Configuring DRIVER

The Web Services DRIVER program can also have a parameter file to control its operation. The default installation does not include this file; if you want to modify the default settings, you must create it. The file must be named `WEBSERVICES/PARAM` and be located under the same usercode and on the same family as the Web Services code files. It can be any kind of text or source file, such as DATA, SEQDATA, or ALGOL.

The syntax for all settings is:

```
<setting> = <value>;
```

Each setting should be on its own line in the file, and the ending semi-colon is required.

Comments can also be included in the file; they must be preceded by a percent sign (%). All characters after the percent sign on the same record will be ignored by the DRIVER program.

The following settings are available (allowed values for settings are shown in curly brackets following the setting name):

**ACCESSCODE = <accesscode>**

The accesscode used when logging on the PSH stations; the default is no accesscode.

**ACCESSKEY = <accesskey>**

The Web Services AccessKey. The Web Services software requires a valid AccessKey to operate. If you do not have an AccessKey, or if your AccessKey has expired, please contact MGS, Inc. See Section 2.4 for more details on host licensing.

**ACPASSWORD = <password>**

The accesscode password to be used when logging on to the PSH stations. The DRIVER program encodes the password and then converts this statement in the parameter file to the *ACPWENCODE* statement (so the password is not maintained in the parameter file as clear-text).

**ALLOW_SELF_SIGNED = { TRUE | FALSE }**

If FALSE, when Web-Out makes an SSL connection and the certificate that is received is self-signed (has a Certificate Authority that points to itself), the connection will be denied. If TRUE, a self-signed certificate will be considered acceptable and the SSL connection will proceed normally. The default value is FALSE.

**ALLOW_UPLOADS = { TRUE | FALSE }**

If FALSE, the DRIVER program will not allow WSUtility to upload new host configuration or include files to the host. The default value is TRUE.

**CCSNUMBER = <integer>**

Use this command to specific a character set number if your MCP system does not use the native MCP EBCDIC character set. Specifying a CCSNUMBER causes MGSWeb to build custom translate tables to convert your outbound application data to UTF-8, and to convert the inbound XML data from UTF-8. The list of valid values can be found under the `EXTMODE` file attribute in the *File Attributes Programming Reference Manual*. Note that not all valid EXTMODE values are supported (no multi-byte characters sets are supported). The default value is 4 (ASERIESEBCDIC).

**ENABLE_RESPONDER = { TRUE | FALSE }**

If TRUE, the DRIVER program will automatically enable the RESPONDER program during initialization, and disable it when the DRIVER program terminates; the default is TRUE.

**ENCODING = <literal>**

This command allows you to specify a custom XML encoding literal; the default is "utf-8". Note that this value is also used in the "Character-Encoding" HTTP attribute when building the outbound XML. Also note that specifying a custom encoding value does not change the way your application data is translated from EBCDIC to ASCII, or the way inbound XML data is translated from ASCII to EBCDIC.

**IDLE_TIMEOUT = { 10..3600 }**

The time, in seconds, of inactivity for a PSH station or database interpretive interface before it is deallocated; the default is 600 seconds (ten minutes).

**MAX_DATABASES = { 1..256 }**

The maximum number of concurrent database interpretive interfaces allowed; the default is 16.

**MAX_STATIONS = { 1..1024 }**

The maximum number of concurrent stations allowed; the default is 256.

**MAX_WORKERS = { 1..16 }**

The number of DRIVER workers active to handle incoming requests; the default is 2.

**PASSWORD = <password>**

The password used to log on to the PSH stations; the default is no password. The DRIVER program encodes the password and then converts this statement in the parameter file to the *PWENCODE* statement (so the password is not maintained in the parameter file as clear-text).

**PREFIX = <station name prefix>**

The prefix for the PSH station names (station names will be "<*prefix*>/<*mix number*>*STAn*", where <*mix number*> is the DRIVER program's mix number and *n* is the station number); the default is "WEBSERVICES".

**PSH = <title>**

The title of the PSH library; the default is WEBSERVICES/PSH.

**REQUEST_TIMEOUT = { 10..300 }**

The time, in seconds, allowed before a request times out; the default is 60 seconds.

**STATION_HEIGHT = { 10..127 }**

The default height, in rows, of the stations allocated by the PSH; the default is 24 rows.

**STATION_WIDTH = { 10..255 }**

The default width, in columns, of the stations allocated by the PSH; the default is 80 columns.

**STREAM_FAMILY = <family name>**

The family the inbound stream files should be stored on; by default, it is the same family the MGSWeb code files are on. Stream files are created under the WEBSERVICES/TEMPFILES/= directory by fields that have the *stream* or *base64Stream* field type.

**TEMPFILES_LIFE = <minutes>**

Stream files received by MGSWeb are stored in the WEBSERVICES/TEMPFILES/= directory, and they are automatically removed after a period of time. This option allows you to control the auto-removal time, which is calculated from the time the receiving application last accessed the file. The default value is 15 minutes. You can set this value to zero to disable the auto-removal of files from the TEMPFILES directory. Note that if an application wishes to keep a received stream file, it should rename the file and/or move the file to another family.

**TRACE = { { <project type> <project name> } ON | OFF }**

The TRACE command allows you to enable host tracing for a specific project or for all projects. If host tracing is enabled for a project, the request and response XML and record data is written to the WEBSERVICES/LOG file.

To enable tracing for a specific project, use the "TRACE <project type> <project name> ON" form of the command. <project type> is either WINDOW (for station projects), PROGRAM, DATABASE, or WEBSERVICE. The <project name> is the name you entered on WSUtility's main "Properties" page for the project.

To enable tracing for all projects, use the "TRACE ON" form of the command.

**USERCODE = <usercode>**

The usercode used to log on to the PSH stations; the default is the ATLAS anonymous usercode, or if the Web Services server is used, the usercode ANONYMOUSWEB.

**USER_LIST = <Windows user account list>**

This feature allows sites to control which users can run WSUtility. Using the new USER_LIST parameter file command, a site can list the Windows user accounts that are allowed to run WSUtility. If the USER_LIST command is not in the parameter file, any user can run WSUtility; if a USER_LIST command is added to the parameter file, only the listed users can run WSUtility.

The syntax of the new command is: "USER_LIST = <Windows user account list>", where <Windows user account list> is one or more Windows user accounts, separated by commas. Each Window user account name is limited to 24 characters, and the list is limited to 20 user accounts.

**WEBOUT_LINK = { TRUE | FALSE }**

If TRUE, the DRIVER program will link to the WS-Out client library to keep it running. By default, the WS-Out client library (`WEBSERVICES/LIBRARY`) terminates when no clients are linked to it.

## 3.4 Configuring the Responder Program

The Program transaction path requires a COMS program to inject the incoming web service transactions into the COMS environment, and to transfer the COMS application responses back to the Web Services software. The RESPONDER program included with the Web Services for Unisys MCP software performs this function.

You only have to configure the RESPONDER program if you are going to implement inbound WS-In Program transactions.

You use the COMS Utility window's PROGRAM screen to configure the RESPONDER direct program; use the following settings on the *Program Activity* screen:

| | |
|---|---|
| *Program Name*: | RESPONDER |
| *TITLE*: | (<usercode>)WEBSERVICES/RESPONDER ON <family> |
| *Usercode*: | Either the usercode the MGSWeb software is running under, or a period if it is running un-usercoded |
| *Maximum Copies*: | 1 |
| *Minimum Copies*: | 1 |
| *Remote-File Interface*: | N |

Note that <usercode> and <family> are the usercode and family you loaded the MGSWeb code files to. The RESPONDER program does not require a window or an agenda.

## 3.5 Data Translation and Character Sets

MGSWeb uses the UTF-8 character set for all XML it generates, and it expects all incoming XML data it receives to be in the UTF-8 character set. By default, MGSWeb uses the standard `EBCDICTOASCII` translate table to convert outbound application data from EBCDIC to UTF-8, and the standard `ASCIITOEBCDIC` translate table to convert inbound XML data from UTF-8 to EBCDIC.

If you are using a character set other than the native MCP EBCDIC character set (ASERIESEBCDIC) on your MCP system, you may have to specify that character set's number via the CCSNUMBER parameter file command, so MGSWeb can correctly convert your application data to UTF-8 and convert incoming UTF-8 data into the character set your system is using.

For certain situations, being able to specify a different encoding literal (other than “UTF-8”) in the XML, and in the HTTP “Character-Encoding” attribute, may be sufficient. You can use the ENCODING parameter file command to specify a different encoding literal. Note that that specifying a custom encoding literal does not change the translation tables used to convert your outbound application data from EBCDIC to ASCII, or the inbound XML data from ASCII to EBCDIC.

Note that you can not specify both a CCSNUMBER value and an ENCODING literal in the parameter file at the same time.

# 4. Starting the Host Software

Once you have completed the configuration required for your site, the DRIVER program must be run.

1. If you are using the MGS Web Services server, start `WEBSERVICES/WFL`. This WFL will run SERVER, which will run DRIVER. The WFL may be started under a usercode or un-usercoded. Your MCP system operation utility or staff must then make sure that this WFL is always running to ensure that your MCP system will continually provide the configured Web Services.

2. If you are using Unisys's Web Transaction Server (ATLAS), it will initiate the DRIVER program when the first request is received for the Web Services virtual directory.

If you have configured the RESPONDER program, and the `ENABLE_RESPONDER` option is TRUE (the default), the DRIVER program will automatically enable the RESPONDER program during initialization.

If you have set the `ENABLE_RESPONDER` option to FALSE, you must manually make sure the RESPONDER program is enabled. Use the COMS command `STATUS PROGRAM RESPONDER` to check the program's status, and the `ENABLE PROGRAM RESPONDER` command to enable it, if necessary.

# 5. Host Software Operation

The SERVER and DRIVER programs, and the PSH library, accept AX commands to control their operation.

They all accept the STOP command (<mix number> AX STOP) to terminate. If you terminate the SERVER program this way, the DRIVER program will also be terminated.

## 5.1 SERVER

The SERVER program only accepts the STOP command, as documented above. Use this command to terminate the Web Services software (including DRIVER).

## 5.2 DRIVER

The DRIVER program also accepts the following AX commands (changes made by these commands will be lost when the DRIVER program terminates – to make permanent changes, you must modify the `WEBSERVICES/PARAM` parameter file). Allowed values follow the command names in curly brackets.

***Note:*** If you are using ATLAS as your web server, you can also enter these commands via the "`NA <provider> <application name> <command>`" ODT command syntax (for example, "`NA ATLASSUPPORT HOSTWEBSERVICES STATUS`").

**`MAX_DATABASES { 1..256 }`**
Sets the maximum number of allowed database connections.

**`MAX_STATIONS { 1..1024 }`**
Sets the maximum number of allowed PSH stations.

**`MAX_WORKERS { 1..16 }`**
Sets the number of DRIVER workers.

**`RELOAD { STATION | PROGRAM | DATABASE | WEBOUT }`**
This command forces a reload of all station, program, database, or WS-Out and HTTP-Out host configuration files. The `RELOAD WEBOUT` variant will also force any persistent outbound TCP/IP connections closed.

**`TRACE { { <project type> <project name> } ON | OFF }`**
The TRACE command allows you to enable host tracing for a specific project or for all projects. If host tracing is enabled for a project, the request and response XML and record data is written to the `WEBSERVICES/LOG` file.

To enable tracing for a specific project, use the "TRACE <project type> <project name> ON" form of the command. <project type> is either WINDOW (for station projects), PROGRAM, DATABASE, or WEBSERVICE. The <project name> is the name you entered on WSUtility's main "Properties" page for the project.

To enable tracing for all projects, use the "TRACE ON" form of the command. "TRACE OFF" will turn off all tracing.

**TERMINATE_WORKERS**

Terminates all workers, and aborts any active requests. The workers are restarted automatically.

**STATUS { DETAIL }**

Displays the current status of the DRIVER program; if you include the DETAIL option, information is displayed on each active station and database.

## 5.3 PSH

The PSH library is the software component that provides the interface between WS-In and a virtual COMS station. It writes information about station allocates/deallocates, any errors, and any trace information to the Web Services log file, called `WEBSERVICES/LOG`. The PSH library accepts the following AX commands (in addition to the STOP command):

**STATUS { <station index> | ALL }**

Displays the current status of a single station, or of all allocated stations (if ALL is specified).

**CLEAR { <station index> | ALL }**

Clears and deallocates a single station, or all stations (if ALL is specified). This will also abort any requests using the stations.

**TRACE { ON | OFF }**

If ON, the PSH tracing option is enabled. This causes PSH activity information to be written to the `WEBSERVICES/LOG` file.

## 5.4 Host Tracing

You can enable the host trace option from within the `WEBSERVICES/PARAM` file, via the TRACE operator command to the DRIVER program, or via WSUtility's **Tracing** menu.

You can enable tracing for a specific project, or for all projects. You can also enable PSH tracing to trace PSH station activity.

When the host trace option is enabled for a project, all transactions to that Station, Program, Database or WebService project cause the request and response XML, and the contents of the COBOL input and output records, to be written to the `WEBSERVICES/LOG` file. All message traffic is included in the tracing information. For example, here's a trace listing that shows a WS-Out transaction:

```
10:53:52 (01739) (MGS)WEBSERVICES/WEBOUT ON WORK v3.03 initializing
10:53:52 (01737) Loading
      (MGS)WEBSERVICES/HOSTCONFIG/WEBSERVICE/CURRENCYCONVERTOR ON WORK.
10:53:52 (01737) Server URL
      "http://www.webservicex.net/CurrencyConvertor.asmx":
      host=www.webservicex.net, port=80,
      directory=/CurrencyConvertor.asmx, hash=4"854B665660E2"
10:53:52 (01737) Field Dump for message ConversionRateSoapIn (3 fields)
  #1 REQ-CONVERSION-RATE parent=0 offset=0 length=20
  #2 REQ-FROM-CURRENCY parent=1 offset=0, usage=1 (length=10)
     0 |USD       |
  #3 REQ-TO-CURRENCY parent=1 offset=10, usage=1 (length=10)
     0 |EUR       |
10:53:52 (01737) HTTP Headers for operation ConversionRate (length=204)
     0 |POST /CurrencyConvertor.asmx HTTP/1.1  Content-Type: text/xm|
    60 |l; charset=utf-8  Content-Length: 376  Host: www.webservicex|
   120 |.net  Connection: close  SOAPAction: "http://www.webserviceX|
   180 |.NET/ConversionRate"    |
10:53:52 (01737) XML Request for operation ConversionRate (length=376)
     0 |<?xml version="1.0" encoding="UTF-8" ?><soap:Envelope xmlns:|
    60 |xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="h|
   120 |ttp://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.|
   180 |xmlsoap.org/soap/envelope/"><soap:Body><ConversionRate xmlns|
   240 |="http://www.webserviceX.NET/"><FromCurrency>USD</FromCurren|
   300 |cy><ToCurrency>EUR</ToCurrency></ConversionRate></soap:Body>|
   360 |</soap:Envelope>|
10:53:53 (01737) XML Response for operation ConversionRate (length=382)
     0 |<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:s|
    60 |oap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="h|
   120 |ttp://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://|
   180 |www.w3.org/2001/XMLSchema"><soap:Body><ConversionRateRespons|
   240 |e xmlns="http://www.webserviceX.NET/"><ConversionRateResult>|
   300 |0.6742</ConversionRateResult></ConversionRateResponse></soap|
   360 |:Body></soap:Envelope>|
10:53:53 (01737) Field Dump for message ConversionRateSoapOut (2 fields)
  #1 RSP-CONVERSION-RATE-RESPONSE parent=0 offset=0 length=6
  #2 RSP-CONVERSION-RATE-RESULT parent=1 offset=0, usage=3, value=0.6742
10:53:53 (01737) Result String: "No error"
10:53:59 (01739) (RWS)WEBSERVICES/WEBOUT ON WORK terminating
```

# 6. Deploying WS-In Web Services

You use the Web Services Utility (WSUtility) to deploy your Web Services.

Deploying a Station transaction Web Service consists of the following steps:

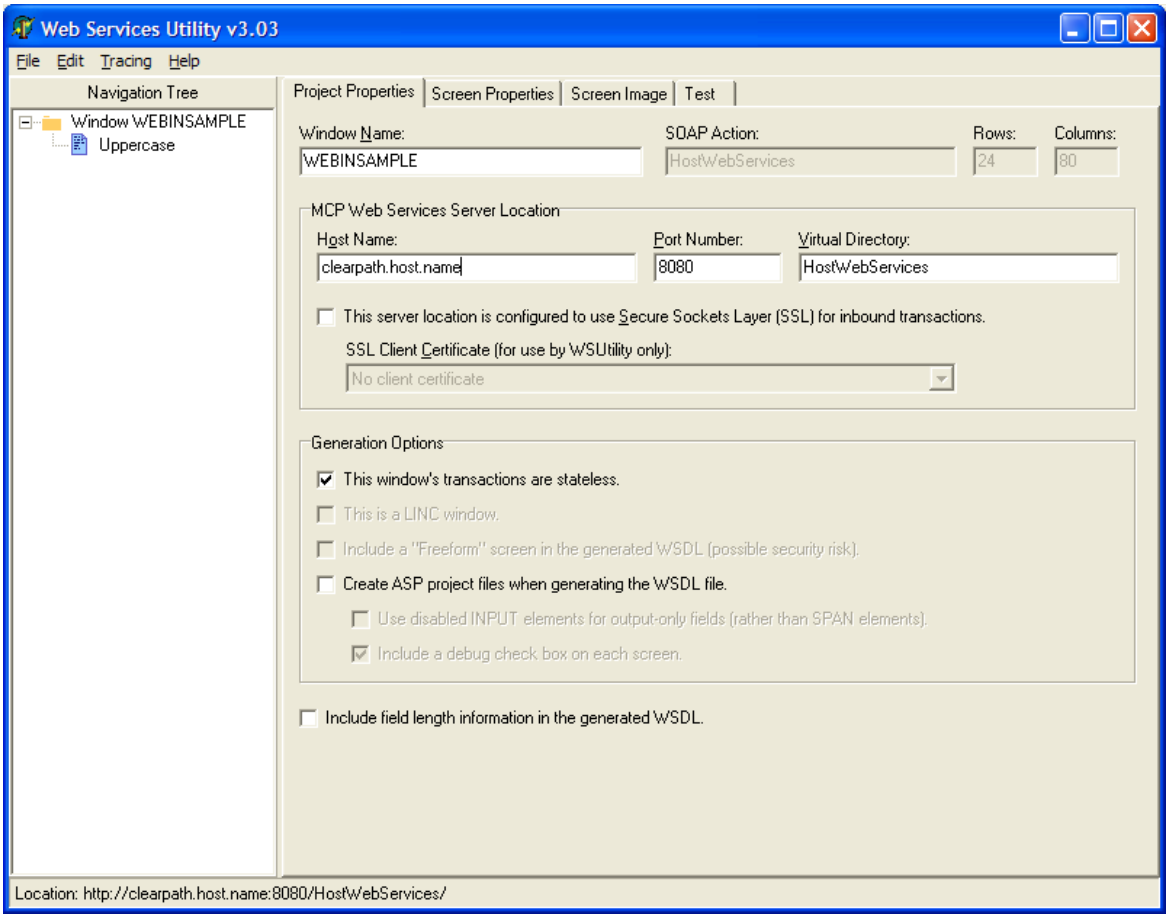
- Running Unisys's Screen Object Model Studio (SOMS) to mark and name the fields and screens you want to make available as a Web Service.
- Running WSUtility to open the SOMS project file.
- Modifying the project settings.
- Optionally testing the web service transaction(s).
- Generating the WSDL and host configuration files.

Deploying a Program transaction Web Service consists of the following steps:

- Creating a new Program project.
- Adding the transactions, including adding request and response record fields for each.
- Optionally testing the web service transaction(s).
- Generating the WSDL and host configuration files.

Deploying a Database transaction Web Service consists of the following steps:

- Creating a new Database project.
- Retrieving the database schema from the host.
- Optionally testing the web service transaction(s).
- Generating the WSDL and host configuration files.




**WSUtility Window**

The left side of the WSUtility window contains a navigation tree, and the right side contains multiple tabbed pages. The contents of the navigation tree and the available tabs depend on the current project type.

## 6.1 Station Transactions

For Station transactions, WSUtility imports a SOMS project file. It gets the names of all screens and fields, the host, and the COMS window from the SOMS project file.

You open a SOMS project file by clicking the **File | Open SOMS project** menu item. You can not create a SOMS project from within WSUtility; you must run SOMS to create a SOMS project.

When a SOMS project is open, the navigation tree will contain a list of screens and any screen groups you have created. The following tabs will be displayed:

| | |
|---|---|
| *Project Properties* | General project settings. |
| *Screen Properties* | Settings for the screen selected in the navigation tree. |
| *Screen Image* | The image of the screen selected in the navigation tree. |
| *Test* | Allows you to test the selected screen. |

### 6.1.1 Creating a SOMS Project

To create the SOMS project file that is used as input for the WS-In Station transactions build process, you have to first create a SOMS project and "capture" each screen that you want to use as a web service. ***Once the "capture" in complete, the SOMS software is no longer used*** and final configuration/deployment is done by WSUtility.

| | |
|---|---|
| ***Note:*** | Before starting, you must install SOMS on your Windows system. SOMS can be found on the Screen Object Modeling Studio CD that comes as part of your Unisys MCP Release CDs. Simply place the SOMS CD in your CD drive and follow the Installer Wizard instructions. |

This section contains a quick overview of using SOMS to capture screens; for more detailed information, see the SOMS documentation.

Once SOMS has been installed and is running, create a new project by selecting the **File | New Project** menu item.

On the *New Project* dialog, select a project type, specify a project name, and click the **OK** button. You may select any of the project types, but we recommend that you use **Microsoft .Net Web Service** as the project type. We also recommend that you use the name of the window from which you are capturing screens as the SOMS project name.

On the *Applications Properties* dialog, enter the name of the host to connect to, and the name of the window you are capturing screens from, and click **Finish**. The MARC logon screen should then be displayed. Once you have entered a valid usercode and password, you will be automatically placed on the host window you specified.

**Page Tabs**

There are at least three tabs at the bottom of the SOMS window (depending on the project type you created, there may be more than three). You will be only using the first two tabs:

| | |
|---|---|
| *Navigate* | This tab displays the terminal emulator page; you use this page to transmit host commands to navigate to the screen(s) you want to capture. Use the **Enter** key to transmit data to the host. |
| *Screen Markup* | This tab displays the current screen, and lets you mark and name fields on the screen. |

To capture a host screen, you first have to use the *Navigate* tab to enter the host commands necessary to display the host screen. Once the screen is displayed, click the *Screen Markup* tab.

**Field Names**

SOMS automatically converts the unprotected fields on the screen to fields with generic names (ie, `inputField_0` for the screen's first field). To rename a field, double-click it to display the *Field Properties* dialog, enter the new name, and click the **OK** button.

**Field Arrays**

Web Services supports screen "field arrays", which allows you to group multiple fields on a screen together using the same array name. You access a field within a field array via the array name and an index.

To create a field array, you must name all of the fields in the array <*array name*>_<*index*>_, where <*array name*> is the same identifier for all fields in the array and <*index*> is the zero-based index for the field in the array.

***Note:*** <*index*> must start with zero and must be consecutive with no missing indices.

For example, to create a three-element field array named myArray, you would name the fields **myArray_0_**, **myArray_1_** and **myArray_2_**.

**Recognition Fields**

You must create one or more *recognition fields* for the screen. The Web Services host software will use the recognition field(s) to identify the screen when it is received from the PSH station, so these fields must contain text that is unique to this screen.

- To select an area on the screen, click the mouse at the start of the field and, holding the mouse button down, move the mouse pointer to the end of the field.

- Mark the selected area as a recognition field by clicking the *Recognition* toolbar button or selecting the **Edit | Set as Recognition Field** menu item.
- If the field is common to all of the screens for this project, select the *Automatically create this field on all new screens* option on the Properties dialog.

**Dynamic Fields**

You can also create *dynamic fields* on the screen, for protected areas of the screen that contain dynamic data (ie, error messages or status information) that you want to have included in the web service response (these are output-only fields).

- Use the mouse to select the area as with the recognition fields.
- Mark the selected area as a dynamic field by clicking the *Dynamic* toolbar button or selecting the **Edit | Set as Dynamic Field** menu item.
- If the field is common to all of the screens for this project, select the *Make this region dynamic on all new screens* option on the Properties dialog, or use the *Multidynamic* toolbar button or the **Edit | Set as Dynamic on all Screens** menu item.

**Saving the Screen**

You can then save the screen by selecting the **File | Save** menu item, entering a screen name, and clicking the **OK** button. This will create the file `<screen name>.xml` in the `<project name>\Screen Definitions\` directory, which contains the captured T27 screen image (ie, the static text on the screen).

After you have captured all of the screens to be made available as Web Services, exit SOMS using the **File | Exit** menu item.

## 6.1.2 Navigation Tree

When you run WSUtility and open a SOMS project, the navigation tree on the left side of the window displays a list of the screens that have been captured by that SOMS project.

The navigation tree also allows you to create *screen groups*, which are used by the ASP project that WSUtility can build when it generates the WSDL file. To create a screen group, you click the **Edit | Add group** menu item. You can then drag the screens and drop them on the screen group to add the screen to the group. Screen groups allow you to group the screens in a logical order, rather than having all screens listed directly under the window name. Screen groups can also contain other screen groups. If you delete a group, all of its screens are moved back directly under the window name in the navigation tree.

### 6.1.3 Project Properties Page

If you are using the MGS Web Services server, the *Port Number* field on the Project Properties page must match the port number in the `WEBSERVICES/SERVER/PARAM` file; the value in the *Virtual Directory* field is ignored.

If you are using the ATLAS server, the *Port Number* field must match the ATLAS server port, and the *Virtual Directory* field must match the virtual directory you created for Web Services with Site Manager. If the ATLAS port is an SSL port, check the *SSL Option* under the *Host Name* field, and optionally select a client certificate from the *SSL Client Certificate* drop-down list (if ATLAS is configured to require certificates).

If the window's transactions are stateless, select the *This window's transactions are stateless* option. If a window's transactions are *stateless*, the application does not save data about the originating station ("state") between transactions, and any transaction can be submitted at any time (that is, no transaction is dependent on the result of previous transactions). (If you do not check this option, client programs will be required to allocate a station before doing any transactions to this window.)

If the window is a LINC window, check the *This is a LINC window* option. This option forces the DRIVER program to discard host responses that do not contain at least one "transmit-protected" field on page 1 of the PSH pseudo station. A window can not be marked as both stateless and a LINC window.

If you want client programs to be able to submit "freeform" transactions to this window, check the *Include a "Freeform" screen in the generated WSDL* option. This option causes WSUtility to insert a screen called "Freeform" in the generated WSDL, which allows client programs to submit any command to the host and get the response in a single field. This option can only be set if the *stateless* option is **not** set.

Optionally, you can have the WSUtility generate an ASP project when it generates the WSDL file. The ASP project can be used to access the Station Web Services you have defined via a web browser without any programming. Check the *Create ASP project files when generating the WSDL file* option to have the WSUtility generate the ASP project. See the **ASP Project** section later in this document for details on using the ASP project.

Normally, WSUtility does not include the length of each element in the WSDL that it generates, because most development environments do not use this information. If you want field lengths stored in the WSDL, you can check the *Include field length information in the generated WSDL* option. Note that enabling this option will make the WSDL larger and more complex.

### 6.1.4 Screen Properties Page

The Screen Properties page lists the fields on the screen selected in the navigation tree. This field list is read-only; to make changes to the fields on the screen, you must re-run SOMS, and then re-import the SOMS project back into WSUtility.

If a field is a member of a field array, the field's index will be displayed in brackets after the field name.

There is also an *Output Screen* field on this page to allow you specify the response screen for the screen selected in the navigation tree. You would need to set this if an input screen receives a different screen as output (by default, Web Services assumes screens receive themselves as output), or if a screen is output-only (that is, a screen that can not be submitted to the system as a web service, but may be returned as a response).

### 6.1.5 Screen Image Page

This page displays the selected screen's image. Regular request fields are displayed in blue, fields in a field array are in green, "transmit protected" fields are gray, and dynamic (output-only) fields are in red. If you place the mouse cursor over a field, the name of the field (and its index, if it is in a field array) will be displayed in a "hint" window.

### 6.1.6 Freeform Restrictions Page

This page is only visible if you have set the *Include a "Freeform" screen in the generated WSDL* option. You can use this page to restrict the text allowed to be submitted to the "Freeform" screen.

To leave the Freeform screen unrestricted, select the *Allow any Freeform screen input* option.

| ***Note:*** If you do not restrict the Freeform screen, client programs will be able to submit any command to the PSH station, which is a potential security risk. |
|---|

To restrict the Freeform screen, select the *Allow only freeform transactions that start with the following literals* option, and then add the allowed transaction literals to the displayed list by entering them in the *New Literal* field and clicking the **Add** button. To delete an entry from the list, select it and click the **Delete** button. With this option set, transactions submitted to the Freeform screen will be accepted only if they start with one of the literals in the list.

### 6.1.7 File Generation

Once you have modified the settings of the project you have open, you can generate the host configuration file, so the host Web Services software can process the web service transactions, and a WSDL file, so client programs can import the WSDL and call the MCP Web Services.

To save your project settings, the WSUtility creates the file `_settings.xml` in the SOMS project directory. The project settings are automatically saved if you generate either a WSDL or host configuration file, or you can click the **File | Save** menu item to manually save the project settings.

**Generating the WSDL File**

You generate the WSDL file by clicking the **File | Generate WSDL file** menu item. If you have the *Create ASP project files* option set, the WSUtility will also generate the ASP project files. The default name for the WSDL file for a Station transaction project is `<window name>.WSDL`.

To make the WSDL file available via the Internet or an intranet, you will have to create the WSDL file in (or copy it to) a web server directory. If you are using ATLAS, you can even put the WSDL file on the MCP host.

**Generating the Host Configuration File**

The WSUtility generates Station host configuration files under the `WEBSERVICES/HOSTCONFIG/WINDOW/=` directory on the host system. These files are sent to the host via the same port that the normal Web Services use (using the MGS Web Services server or ATLAS). After the files are transferred to the host, the DRIVER program reloads them so new settings are automatically picked up.

Host configuration files contain the information necessary for the DRIVER program to convert the incoming Web Services request from XML to a screen image. The files are in XML format, and are EBCDIC stream files with a record size of one byte.

To generate the host configuration file, you click the **File | Generate host config file** menu item. You must have the host server software (either the MGS Web Services server or the ATLAS server) configured and running on the host before doing this, because the DRIVER program receives the host configuration information and creates the file on the MCP system.

When you click the **Generate host config file** menu item, a transfer dialog is displayed, reminding you that the host server must be running. When it is, click the **Start** button. The data will then be transferred automatically, and the name of the new host configuration file will be displayed. If an error occurs, details will be displayed on the transfer dialog.

### 6.1.8 Testing

Once you have generated the host configuration file, you can use the *Test* page to submit test transactions to verify each Station transaction.

The *Test* page contains two tabs at the bottom of the page – *Request Data*, which lists the request fields for the transaction, and *Response Data*, which displays the response's fields. The Test page also contains a **Submit** button to send the transaction to the host (this button is only enabled when the *Request Data* tab is selected), and an **Allocate** button (used to allocate a station on the host so you can test stateful transactions). If you have allocated a station, the **Allocate** button will become the **Deallocate** button.

To test a particular screen's transaction, select it in the Navigation tree, and click the *Test* tab. Click the *Request Data* tab, enter the test data into the fields on the screen, and click the **Submit** button. The *Response Data* tab will be automatically selected when the response is received so the response data is visible.

## 6.2 Program Transactions

When you create a new Program project, or open an existing one, WSUtility allows you to create *transactions*. A transaction consists of a transaction name and a request and a response record. Each request and response record consists of MCP Header fields, optional SOAP Header fields, and one or more SOAP Body fields.

When a Program project is open, the navigation tree will contain a list of transactions you have created, and the following tabs will be displayed:

| | |
|---|---|
| *Program Properties* | General project settings. |
| *Transaction Fields* | Lists the fields for the selected transaction. |
| *Test* | Allows you to test the selected transaction. |

### 6.2.1 Navigation Tree

When a Program project is open, the navigation tree lists the transactions in the project. The first transaction in a new Program project is created for you automatically as *newTransaction_0*. You can use the **Edit** | **Rename transaction** menu item to rename existing transactions, and the **Edit | Add transaction** menu item to create more transactions.

### 6.2.2 Program Properties Page

The Program Properties page lets you set the program name, host name, port number, virtual directory, and the MCP header SSL options for the Program project. The program name you enter here should match the application program that will be handling the transactions.

If you are using the Web Services server, the port number on this page must match the port number in the `WEBSERVICES/SERVER/PARAM` file; the value in the *Virtual Directory* field is not used.

If you are using the ATLAS server, the *Port Number* field must match the ATLAS server port, and the *Virtual Directory* field must match the virtual directory you created for Web Services with Site Manager. If the ATLAS port is an SSL port, check the *SSL Option* under the *Host Name* field, and optionally select a client certificate from the *SSL Client Certificate* drop-down list.

The MCP header fields are items included in the request records by the Web Services software to help your application program identify which transaction is being requested, and also to identify the client calling the Program transaction.

By default, the request record's MCP header fields include a six-byte transaction token, the name of the transaction, and the remote client IP address, and the response record contains the transaction token. If you have configured ATLAS to require two-way SSL, you can use the *SSL Client Certificate Transaction MCP Header Fields* options on the Properties page to have information about the client's certificate included in the MCP header fields. Please see the **Accessing WS-In Web Services from Other Systems** section later in this document for more details.

Normally, WSUtility does not include the length of each element in the WSDL that it generates, because most development environments do not use this information. If you want field lengths stored in the WSDL, you can check the *Include field length information in the generated WSDL* option. Note that enabling this option will make the WSDL larger and more complex.

### 6.2.3 Transaction Properties Page

You use this page to add, rename, and delete fields in both the request and response records for the transaction selected in the navigation tree. You use the tabs at the bottom of the *Transaction Properties* page to select either the request record or the response record. The selected record is displayed in the list box in the middle of the page in a COBOL "01" record format.

Each record consists of three sections: the *MCP Header*, the *SOAP Header*, and the *SOAP Body*. You can not modify the MCP Header fields. You can add fields to either the SOAP Header or SOAP Body of a record. Normally, you would add all fields to the SOAP Body, unless a client application requires SOAP Header fields.

You add fields by selecting either the *SOAP Header Fields* or *SOAP Body Fields* item in the field list, entering the new field's XML name, COBOL name, type, length, and occurs value in the fields above the record display, and clicking the **Add** button. If you do not enter a COBOL name, a default one will be generated from the XML name. You can also use the *Field Parent* drop down list to select the parent field for the new field. If you select a non-group field as the field's parent, the new field will be inserted immediately above the selected field in the record.

To modify a field, select it with the mouse, edit the field's XML name, parent, COBOL name, type, length and/or occurs values, and click the **Modify** button. To delete a field, select it and click the **Delete** button.

To create a field array, enter a value greater than one in the *Occurs* field. If a field in either the request or response record is an array, the number of occurrences will be displayed with an "OCCURS" clause after the field name.

You can also create a "group" field; group fields have one or more child fields, which can in turn be group fields. Group field arrays are also supported.

You can move a field within a record by dragging and dropping the field to a new location (including between the SOAP Header and SOAP Body). You can not drag a field between records or transactions, however.

### 6.2.4 Special Field Types

In addition to the standard *group*, *alpha* and *numeric* field types, fields in the request record can also be one of the following special types:

- **base64**: The field will contain data encoded with the base64 encoding algorithm. The DRIVER program will automatically convert the data from base64 to binary in the request record, and from binary to base64 in the response record. This field type allows your application to send and receive binary data. The field length is limited to less than 64K bytes.

- **stream**: This field type allows your application to send and receive large amounts of text data. For "stream" fields in the request record, all data for the field in the inbound XML will be translated from ASCII to EBCDIC, and then written directly to a temporary stream file on the MCP system; the title of the temporary file will be inserted into the request record passed to your application. For "stream" fields in the response record, the application must insert the name of a MCP stream file into the field; MGSWeb will then insert the file's contents (translated from EBCDIC to ASCII) into the outbound XML. "stream" fields are 100 characters long in the request and response records (long enough to contain a file title), but the data in the XML can be of any length.

- **base64Stream**: This type is a combination of the **base64** and **stream** types; it allows your application to send and receive large amounts of binary data. For "base64Stream" fields in the request record, the inbound XML data for the field is converted from base64 to binary and written directly to a temporary stream file; the title of the temporary file will be inserted into the request record passed to your application. For "base64Stream" fields in the response record, the application must insert the title of an MCP stream file in the field; MGSWeb will then insert the contents of the file, converted to base64, into the outbound XML. "base64Stream" fields are 100 characters long in the request and response records (long enough to contain a file title), but the data in the XML can be of any length.

Note that the inbound "stream" and "base64Stream" files will be created under the same usercode as the MGSWeb code files, in the `WEBSERVICES/TEMPFILES/=` directory, with the EXTMODE attribute set to OCTETSTRING.

By default, they will be stored on the same family as the MGSWeb code files; you can change this using the **`STREAM_FAMILY`** setting in the `WEBSERVICES/PARAM` file.

MGSWeb will remove the files in the TEMPFILES directory fifteen minutes after they are last accessed by the receiving application; you can control the auto-removal time using the **TEMPFILES_LIFE** setting in the WEBSERVICES/PARAM file.

## 6.2.5 File Generation

Once you have modified the settings of the project you have open, you can generate the host configuration file, so the host Web Services software can process the web service transactions, and a WSDL file, so client programs can import the WSDL and call the MCP Web Services.

You can also optionally generate a COBOL copy library, to use in your COMS application that processes the transaction in the Program project.

To save your project settings, the WSUtility creates the file <program name>.psf in the directory of your choice. The project settings are automatically saved if you generate either a WSDL or host configuration file. You can also click the **File | Save** menu item to manually save the project settings.

**Generating the WSDL File**

You generate the WSDL file by clicking the **File | Generate WSDL file** menu item. The default name for the WSDL file for a Program transaction project is <program name>.WSDL.

To make the WSDL file available via the Internet or an intranet, you will have to create the WSDL file in (or copy it to) a web server directory. If you are using ATLAS, you can even put the WSDL file on the MCP host.

**Generating the Host Configuration File**

The WSUtility generates program host configuration files under the WEBSERVICES/HOSTCONFIG/PROGRAM/= directory on the host system. These files are sent to the host via the same port that the normal Web Services use (using the Web Services server or ATLAS). After the files are transferred to the host, the DRIVER program reloads them so new settings are automatically picked up.

Host configuration files contain the information necessary for the DRIVER program to convert the incoming Web Services request from XML to a request record. The files are in XML format, and are EBCDIC stream files with a record size of one byte.

To generate the host configuration file, you click the **File | Generate host config file** menu item. You must have the host server software (either the Web Services server or the ATLAS server) configured and running on the host before doing this, because the DRIVER program receives the host configuration information and creates the file on the MCP system.

When you click the **Generate host config file** menu item, a transfer dialog is displayed, reminding you that the host server must be running. When it is, click the **Start** button. The data will then be transferred automatically, and the name of the new host configuration file will be displayed. If an error occurs, details will be displayed on the transfer dialog.

**Generating the COBOL Copy Library**

WSUtility can also generate a COBOL "COPY library" (include file) for use by the COMS application that will process the transactions in the Program project. The include file is named `WEBSERVICES/USERFILES/PROGRAM/<program name>`, under the same usercode and on the same family as the Web Services code files.

The generated include file will contain all request and response records for all transactions defined in the project. All request records will use the same memory area (all but the first one will be a "REDEFINES") so your program can do a COMS "RECEIVE" into any one of them, and not have to copy the data.

To generate the include file, click the **File | Generate include file** menu item. You must have the host server software (either the Web Services server or the ATLAS server) configured and running on the host before doing this, because the DRIVER program receives the host configuration information and creates the file on the MCP system.

When you click the **Generate include file** menu item, a transfer dialog is displayed, reminding you that the host server must be running. When it is, click the **Start** button. The data will then be transferred automatically, and the name of the new include file will be displayed. If an error occurs, details will be displayed on the transfer dialog.

## 6.2.6 Testing

Once you have generated the host configuration file, you can use the *Test* page to submit test transactions to verify each Program transaction.

The *Test* page contains two tabs at the bottom of the page – *Request Data*, which lists the request fields for the transaction, and *Response Data*, which displays the response's fields. The Test page also contains a **Submit** button to send the transaction to the host (this button is only enabled when the *Request Data* tab is selected).

To test a particular Program transaction, select it in the Navigation tree, and click the *Test* tab. Click the *Request Data* tab, enter the test data into the fields on the screen, and click the **Submit** button. The *Response Data* tab will be automatically selected when the response is received so the response data is visible.

## 6.3 Database Transactions

For Database transactions, WSUtility downloads the database's schema from the host. You must specify the name of the host and the title of the DMINTERPRETER library on the Database Properties page before downloading the schema.

**Note:** Visibility of the database (ie, the security of the control file and the DMSUPPORT and DMINTERPRETER libraries) must be set correctly to allow the Web Services software to successfully open a database.

When a Database project is open, the navigation tree will contain a list of the database's datasets, and the following tabs will be displayed:

| | |
|---|---|
| *Database Properties* | General project settings. |
| *Dataset Properties* | Lists the fields in the selected dataset. |
| *Test* | Allows you to test the selected dataset. |

### 6.3.1 Navigation Tree

When a Database project is open, the navigation tree lists the database's datasets after you have retrieved the database's schema from the host.

### 6.3.2 Database Properties Page

The Database Properties page lets you set the host name, port number, virtual directory, and DMINTERPRETER title for the Database project. The database's name is set when you download the database's schema.

If you are using the Web Services server, the port number on this page must match the port number in the `WEBSERVICES/SERVER/PARAM` file; the value in the *Virtual Directory* field is not used.

If you are using the ATLAS server, the *Port Number* field must match the ATLAS server port, and the *Virtual Directory* field must match the virtual directory you created for Web Services with Site Manager. If the ATLAS port is an SSL port, check the *SSL Option* under the *Host Name* field.

Normally, WSUtility does not include the length of each element in the WSDL that it generates, because most development environments do not use this information. If you want field lengths stored in the WSDL, you can check the *Include field length information in the generated WSDL* option. Note that enabling this option will make the WSDL larger and more complex.

After you have set the host name and DMINTERPRETER title, you can click the *Get Schema* button on this page to request a list of the database's datasets and fields. You must make sure the server is configured and running on the host before doing this.

**Dataset Field Limitations**

There are some limitations on dataset fields:

- Group names are discarded; only actual fields are included.
- Any field that has multiple subscripts is discarded.
- Only key fields are included in the request record; however, all fields are included in the response record.

## 6.3.3 Dataset Properties Page

This page is displayed automatically when you select a dataset in the Navigation Tree. It shows the dataset's fields, including the name, type, and length of the field. If the field is a "key" field (it's either a major or minor key in a set), a yellow key will be displayed to the left of its name. If the field is an array of items (it has an OCCURS clause in the DASDL), the number of occurrences will be shown in brackets after the name.

You can also use this page to disable access to the dataset from Web Services; that is, if you un-check the *Allow access to this dataset via Web Services requests* checkbox, the dataset will not be included in the generated WSDL.

## 6.3.4 File Generation

Once you have modified the settings of the project you have open, you can generate the host configuration file, so the host Web Services software can process the web service transactions, and a WSDL file, so client programs can import the WSDL and call the MCP Web Services.

To save your project settings, the WSUtility creates the file `<database name>.dsf` in the directory of your choice. The project settings are automatically saved if you generate either a WSDL or host configuration file, or you can click the **File | Save** menu item to manually save the project settings.

**Generating the WSDL File**

You generate the WSDL file by clicking the **File | Generate WSDL file** menu item. The default name for the WSDL file for a Database transaction project is `<database name>.WSDL`.

To make the WSDL file available via the Internet or an intranet, you will have to create the WSDL file in (or copy it to) a web server directory. If you are using ATLAS, you can even put the WSDL file on the MCP host.

**Generating the Host Configuration File**

The WSUtility generates database host configuration files under the `WEBSERVICES/HOSTCONFIG/DATABASE/=` directory on the host system. These files are sent to the host via the same port that the normal Web Services use (using the Web Services server or ATLAS). After the files are transferred to the host, the DRIVER program reloads them so new settings are automatically picked up.

Host configuration files contain the information necessary for the DRIVER program to convert the incoming Web Services request from XML to a DMSII interpretive interface request. The files are in XML format, and are EBCDIC stream files with a record size of one byte.

To generate the host configuration file, you click the **File | Generate host config file** menu item. You must have the host server software (either the Web Services server or the ATLAS server) configured and running on the host before doing this, because the DRIVER program receives the host configuration information and creates the file on the MCP system.

When you click the **Generate host config file** menu item, a transfer dialog is displayed, reminding you that the host server must be running. When it is, click the **Start** button. The data will then be transferred automatically, and the name of the new host configuration file will be displayed. If an error occurs, details will be displayed on the transfer dialog.

## 6.3.5 Testing

Once you have generated the host configuration file, you can use the *Test* page to submit test transactions to verify each Database transaction.

The *Test* page contains two tabs at the bottom of the page – *Request Data*, which lists the request fields for the transaction, and *Response Data*, which displays the response's fields. The Test page also contains a **Submit** button to send the transaction to the host (this button is only enabled when the *Request Data* tab is selected), and an **Allocate** button (used to allocate a database on the host). If you have allocated a database, the **Allocate** button will become the **Deallocate** button.

To test a particular Database transaction, select it in the Navigation tree, and click the *Test* tab. Click the *Request Data* tab, enter the test data into the fields on the screen, and click the **Submit** button. The *Response Data* tab will be automatically selected when the response is received so the response data is visible.

# 7. Additional WSUtility Features

## 7.1 Host Trace Control

You can control the host trace option for the current project from within WSUtility via the **Tracing** menu. You can retrieve the current project's tracing status from the host, and enable/disable tracing for the current project from this menu. Trace information is written to the Web Services log file (called `WEBSERVICES/LOG`). A project must be open, and a host name and port number specified, for the functions on the **Tracing** menu to work.

## 7.2 ASP Project Details

The WSUtility program can generate an Active Server Pages (ASP) project when you generate the WSDL file for a Station transaction project. The ASP project files are written to the same directory you store the WSDL file in. The generated ASP project provides a complete browser-based solution to use the Web Services you have defined with the SOMS project without any coding.

The ASP project should work with recent versions of both Internet Explorer and Netscape Navigator.

**ASP Files**

The following files are generated by WSUtility:

```
Allocate.html
Deallocate.html
Error1.html
Error2.html
Error3.html
Fault.html
Folder.gif
Logoff.html
Screen.gif
Webservice.asp
Webservice.css
Webservice.js
Welcome.html

<window name>\_menu.html
<window name>\<screen name>.html
```

You can edit any of the `<screen name>.html` files to customize the look of the host screens.

The `_menu.html` file contains the navigation menu for the window, including any screen groups you have created. The ASP project will automatically merge all `_menu.html` files together at runtime to create a single navigation tree.

For example, if you place the ASP project for window TEST1 under the MYHOST directory, and then you also place the ASP project for window TEST2 under the same MYHOST directory, the _menu.html files for TEST1 and for TEST2 will be merged together and displayed in the browser as a single navigation tree.

To access the ASP project from a browser, enter the URL:

```
http://<server>/<directory>/webservice.asp
```

where `<server>` is the name of the IIS web server you installed the ASP project on, and `<directory>` is the virtual directory that contains the ASP project files.

**ASP Generation Options**

There are two generation options available for the ASP project within the WSUtility:

*Use disabled INPUT elements for output-only fields*

If you select this option, output-only fields will be placed on the web page in disabled INPUT elements, rather than in SPAN elements. Some browsers do not support SPAN elements completely; select this option if the output-only fields do not display correctly.

*Include a debug check box on each screen*

If you select this option, each screen will contain, in the lower-right hand corner, an unlabelled checkbox. If the user checks the box before submitting the screen, the XML request and response will be added to the browser response page.

# 8. Accessing WS-In Web Services from Other Systems

Once you have the Web Services software installed and configured on the host, and the host configuration and WSDL files generated, client programs can call the Web Services to access COMS transactions.

These client programs can be written in any language that provides a Web Services interface, such as Visual Basic .NET and Delphi. These development environments can import the WSDL file WSUtility generates, and automatically generate the necessary objects and interfaces to allow you to access the Web Services.

The following sub-sections contain information you may require to access WS-In web services from your client programs.

## 8.1 Program Transaction MCP Header

In order for the Web Services software to match responses to requests for Program transactions, and to help the COMS application processing the transactions to identity both which transaction is being requested and the client, an MCP Header is inserted in front of the request and response records for all Program transactions.

### 8.1.1 Request Record MCP Header Fields

The request header consists of the following fields (the length of the field in bytes follows the name of the field):

**TRANSACTION-TOKEN (6)**
: This field identifies the original request for the RESPONDER program, so the response can be matched to the request.

**TRANSACTION-ID (6)**
: This numeric field contains the ID of the transaction submitted by the client (the ID value for each transaction is set using WSUtility).

**TRANSACTION-NAME (36)**
: This field contains the name of the transaction being submitted, with blank fill. The transaction name will be truncated in this field if it is longer than 36 characters.

**REMOTE-IP-ADDRESS (42)**
: This field contains the IP address of the client that originated the transaction, with blank fill.

If you are using two-way SSL and the ATLAS web server, you can optionally have the Web Services software include client certificate information in the MCP Header. The following optional client certificate fields are available:

**SUBJECT-COMMON-NAME (50)**
This field will contain the common name (CN) from the client certificate's "Subject" distinguished name.

**SUBJECT-DISTINGUISHED-NAME (200)**
This field will contain the client certificate's entire "Subject" distinguished name.

**ISSUER-COMMON-NAME (50)**
This field will contain the common name (CN) from the client certificate's "Issuer" distinguished name.

**ISSUER-DISTINGUISHED-NAME (200)**
This field will contain the client certificate's entire "Issuer" distinguished name.

**PUBLIC-KEY-STRENGTH (6)**
This field will contain the client certificate's public key strength, in bits.

### 8.1.2 Response Record MCP Header Field

The response record's MCP Header consists of only the six-byte `TRANSACTION-TOKEN`.

### 8.1.3 Important Notes

- The COMS application that processes the Program transaction **must** move the `TRANSACTION-TOKEN` header field from the request record to the response record before sending the transaction response, or the response will never be delivered to the client.

- If you enable the client certificate MCP Header options, but you are either using the MGS Web Server, ATLAS is not using SSL, or ATLAS is not configured to request a client certificate, the client certificate MCP Header fields will be blank.

- A "distinguished name" consists of a list of "name=value" pairs, where "name" can be CN (common name), OU (organizational unit), O (organization), L (location), ST (state), or C (country). There are other possibilities for "name" but these are the most common.

## 8.2 Stateless and Stateful Transactions

COMS applications can implement their transactions as *stateless* or *stateful*.

- *Stateless* transactions are not dependent on previous transactions; they can be entered at any time. The data submitted with the transaction contains all of the information necessary for the application to process it; the application does not store any station-specific data between transactions. Many COMS Trancode Based Routing (TBR) applications are stateless – any trancode can be entered at any time.

- *Stateful* transactions require the COMS application to remember data between transactions (the data saved is called "state"). State is usually tied to the originating station – that is, the application stores the state by station. Multi-screen transactions (those that require the user to transmit more than one page of data) are stateful – for instance, the user could not transmit the third page of data before transmitting the first page.

Some applications implement both stateless and stateful transactions.

### 8.2.1 Station Transactions

Web Services' Station transactions can be either stateless or stateful.

The normal submission of a Station transaction via Web Services is stateless – there is a pool of PSH stations available, and the Web Services DRIVER program selects the first available station to process each incoming transaction.

However, Web Services also supports stateful transactions, by allowing a client program to allocate a station for that client's exclusive use. The client is given a station *token*, which is then included by the client in each transaction. The client's transactions (and only that client's transactions) are then all done against the same station. When the client is done with the station, it is deallocated, and returned to the available pool.

A client program can actually do both stateless and stateful transactions at the same time. Even if a station has been allocated, the client can still submit a stateless transaction to the pool of available stations by not specifying the station token in the transaction.

**Transmit-Protected Fields**

Some MCP applications (such as LINC) use "transmit-protected" fields on their screens to implement state. Transmit-protected fields are protected in forms mode, but are transmitted back to the host.

The Web Services PSH handles transmit-protected fields automatically for allocated stations. That is, the PSH saves the data from the transmit-protected fields from each application response and reinserts the data into the next request automatically. (For stateless transactions, the PSH fills all transmit-protected fields with spaces.)

### 8.2.2 Program Transactions

Web Services does not natively support stateful Program transactions; however, either the IP address of the caller, a client certificate field, or a user token field (added to the request and response records) could be used, so the COMS application processing the transaction could identify the client program (and maintain state).

### 8.2.3 Database Transactions

Web Services' Database transactions are all stateful. A client program must allocate a database interpretive interface for its exclusive use. The client is given a database *token*, which is then included by the client in each transaction. When the client is done with the database, it must be deallocated, and returned to the available pool.

## 8.3 WSDL Details

The following WSDL terms are used in this document:

- *operation*, which is a WSDL name for a procedure you can call via Web Services.
- *type*, which is a record, containing either the request or response fields for an operation.
- *port*, which is a list of available operations for a given COMS window or program, or DMSII database.
- *binding*, which ties the port to a transport (HTTP and SOAP).
- *service*, which is the combination of a binding and a host's URL.

WSUtility generates one operation and two types for each screen, transaction, and dataset:

- The operation is named `<screen name>` for Station transactions, `<transaction name>` for Program transactions, and `<dataset name>` for Database transactions. For example, if you capture a screen in SOMS and name it `ScreenOne`, the WSDL operation would also be named `ScreenOne`.

- The two types are named `<operation>Request` and `<operation>Response`; the first type contains the request fields, and the second type contains the response fields for the operation. For example, the two types for `ScreenOne` would be named `ScreenOneRequest` and `ScreenOneResponse`. For the dataset request type, only key fields are included in the request type.

For Program transactions, WSUtility may also generate types for the SOAP Headers and any group fields you have defined in the request or response records. The SOAP Header types will be named `<transaction name>RequestHeader` (for the request header) and `<transaction name>ResponseHeader` (for the response header). The type created for a group field will be named the same as the field itself (although the name may be modified slightly to prevent any duplicate type names).

Note that if you enable the *Include field length information in the generated WSDL* option, WSUtility will also generate types for each field of a different length.

The WSDL also contains a port for the window, program, or database named `Window<window name>Port` for Station transactions, `Program<program name>Port` for Program transactions, and `Database<database name>Port` for Database transactions. For example, for a window named TEST, the port would be named `WindowTESTPort`.

The service declared in the WSDL is called `Window<window name>Service`, `Program<program name>Service`, or `Database<database name>Service`, depending on the project type.

The operations in the WSDL file use the **document/literal** binding style.

### 8.3.1 Allocate and Deallocate Operations

All Station and Database transaction WSDL files also contain two additional operations: `Allocate` and `Deallocate`. These operations can be used when a client needs to allocate a PSH station or database interpretive interface for its exclusive use, for stateful transactions.

**Station Types**

The Station `Allocate` operation has two types defined for it:

- `AllocateRequest` contains the request parameters for the `Allocate` operation:

| | |
|---|---|
| `usercode` | The usercode for the PSH station; if not specified, the usercode/password from the parameter file will be used. |
| `password` | The password for the PSH station; if not specified, a blank password will be assumed. |
| `accesscode` | The accesscode for the PSH station; if not specified, the default accesscode/acpassword from the parameter file will be used (if required). |
| `acpassword` | The accesscode password; if not specified, a blank password will be used. |
| `width` | The width of the station in columns; the value from the parameter file will be used if this field is not specified. |
| `height` | The height of the station in rows; the value from the parameter file will be used if this field is not specified. |

- `AllocateResponse` contains the response fields for the `Allocate` operation:

| | |
|---|---|
| `mcpElapsed` | The elapsed time, in milliseconds, from when the Web Services software received the transaction until the response was sent. This time does not include time spent in the Web Services or ATLAS server. |
| `mcpStation` | The allocated station's token. |

The *Deallocate* operation also has two types defined for it:

- `DeallocateRequest`, which contains only a `mcpStation` field.
- `DeallocateResponse`, which contains the `mcpElapsed` and `mcpStation` fields, although it always returns a value of zero in the mcpStation field.

**Database Types**

The Database `Allocate` operation has two types defined for it:

- `AllocateRequest` contains the request parameters for the `Allocate` operation:
  `usercode` This field is not currently used.
  `password` This field is not currently used.

- `AllocateResponse` contains the response fields for the `Allocate` operation:
  `mcpElapsed` The elapsed time, in milliseconds, from when the Web Services software received the transaction until the response was sent. This time does not include time spent in the Web Services or ATLAS server.
  `mcpDatabase` The allocated database's token.

The `Deallocate` operation also has two types defined for it:

- `DeallocateRequest`, which contains only a `mcpDatabase` field.
- `DeallocateResponse`, which contains the `mcpElapsed` and `mcpDatabase` fields, although it always returns a value of zero in the `mcpDatabase` field.

## 8.3.2 mcpElapsed, mcpStation, and mcpDatabase Fields

**Station Transactions**

All `<operation>Request` types defined in a Station transaction WSDL file contain a `mcpStation` field; you set this field to the token returned in the `AllocateResponse` record to submit transactions to your allocated station. If you do not specify a `mcpStation` value or set it to zero, the transaction is sent to any available PSH station.

All `<operation>Response` types defined in a Station transaction WSDL file contain both a `mcpElapsed` and a `mcpStation` field.

**Program Transactions**

All Program transaction response types contain the `mcpElapsed` field.

**Database Transactions**

All `<operation>Request` types defined in a Database transaction WSDL file contain a `mcpDatabase` field; you must set this field to the token returned in the `AllocateResponse` record to submit transactions to your allocated database.

All `<operation>Response` types defined in a Database transaction WSDL file contain both a `mcpElapsed` and a `mcpDatabase` field.

# 9. Configuring WS-Out and HTTP-Out Web Services

Web Services for Unisys MCP supports two types of outbound web service types: WSDL-based Web Services (which are called a "Web Service," or WS-Out), and non-WSDL-based Web Services (which are called an "HTTP Service," or HTTP-Out).

Enabling an application on the Unisys MCP system to call a WSDL-based Web Service requires the following steps using the WSUtility program:

- Create a new Web Service project.
- Load the web service's WSDL file.
- Make any required changes to the request or response record descriptions.
- Generate the host configuration file.
- Generate a COBOL "COPY library" (include file) for the web service.

Enabling an application on the Unisys MCP system to call a non-WSDL-based web service requires the following steps using the WSUtility program:

- Create a new HTTP Service project.
- Define the request and response records for each operation.
- Generate the host configuration file.
- Generate a COBOL "COPY library" (include file) for the web service.

You can then add code to call the Web Service to your application.

The following sections describe these steps in more detail.

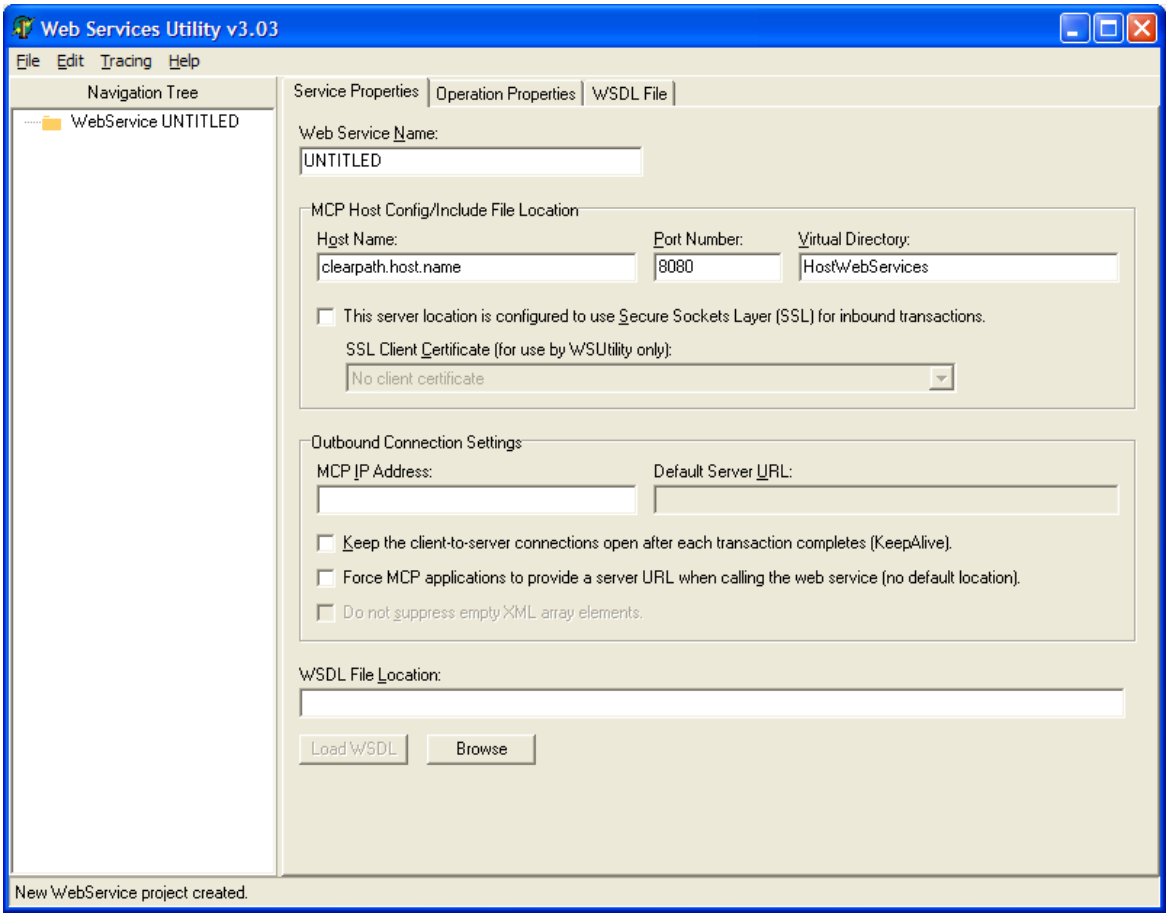
## 9.1 Running WSUtility

To run the Web Services Utility (WSUtility) program, navigate to the folder you installed the Web Services PC software to, and double-click the `WSUtility.exe` file. You can add a shortcut to your desktop for WSUtility by right-clicking on the `WSUtility.exe` file, and clicking the **Send To | Desktop** menu item.

## 9.2 WSDL-Based Web Services

To create a new WSDL-based web service ("Web Service") project, click the **File | New | Web Service project** menu item. To open an existing Web Service project, click the **File | Open | Web Services project** menu item. Web Service project files have a ".wsf" file extension.

After you have opened or created a new Web Service project, the WSUtility window will change to look like this:

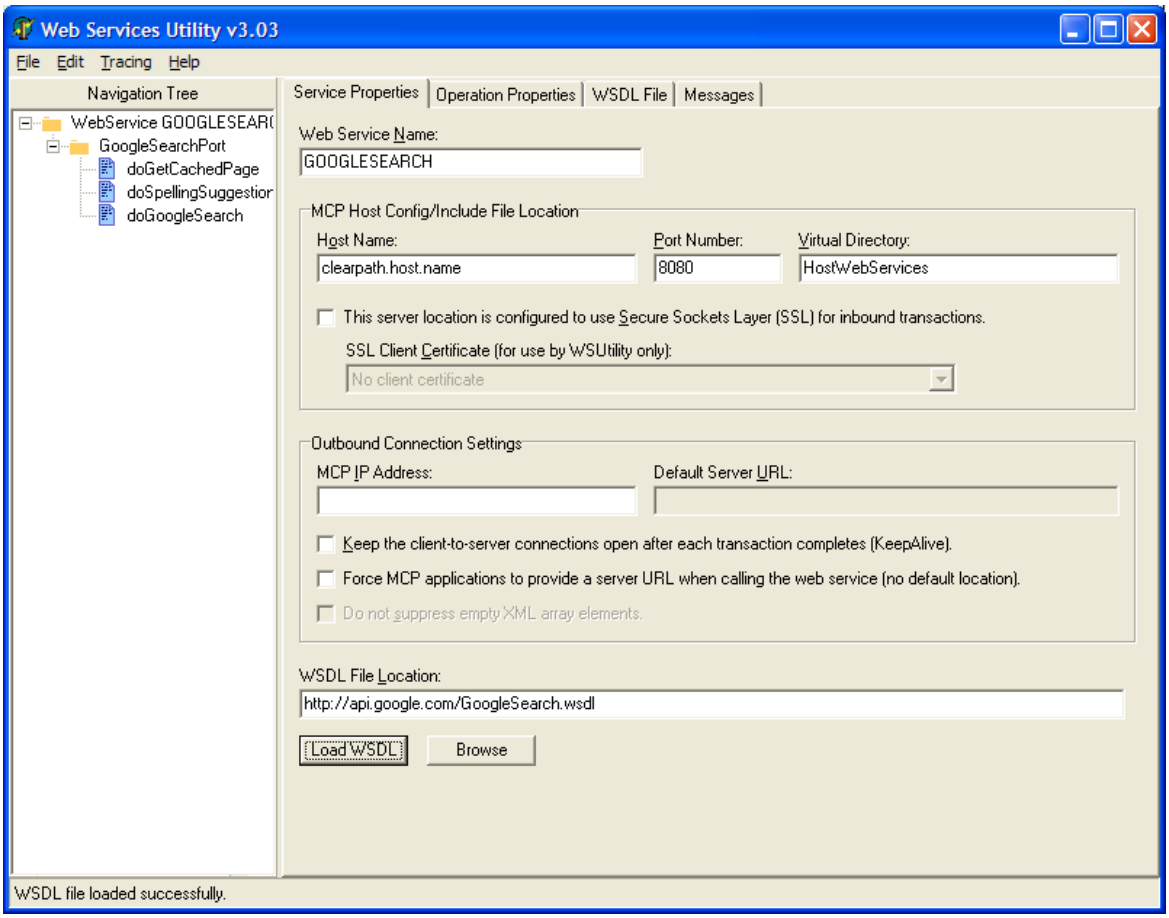



**The WSUtility Web Services Project window**

## 9.2.1 Loading the WSDL File

To load the web service's WSDL file, enter its URL in the *WSDL File Location* field, and click the *Load WSDL* button. If the WSDL file is on your local hard drive, you can click the "browse" button to the right of the location field to insert the name of the WSDL file in the location field.

If the WSDL file load is successful, the ports and operations from the WSDL file will be displayed in the *Navigation Tree* control on the left side of the WSUtility window, as shown below.




**WSUtility with a WSDL loaded**

If one or more errors occur while loading the WSDL file, click on the *Messages* tab at the top of the WSUtility window (the *Messages* tab will be added when the WSDL file load begins) to see a list of the loading status and error messages. Errors will be displayed in red.

On the image shown above (the GOOGLE web service), the web service is called "GOOGLESEARCH", there is a single port named "GoogleSearchPort", and there are three operations, called "doGetCachedPage", "doSpellingSuggestion", and "doGoogleSearch".

## 9.2.2 Service Properties

The name displayed for the Web Service is the name from the WSDL "definitions" node, or the name of the first service defined in the WSDL. WSUtility will convert the name to uppercase letters. Web Service names are limited to 17 uppercase characters - if the name from the WSDL file is too long, you must shorten it to 17 or fewer characters.

**MCP Host Config/Include File Location**

You have to enter the DNS name or IP address, port number, virtual directory and SSL option (and optionally a client certificate name) for the ATLAS or Web Services server on your MCP system in the *MCP Host Config/Include File Location* fields. These are needed so the WSUtility program can transfer the host configuration and COBOL "COPY" files to the host.

**Outbound Connection Settings**

If you enter an IP address in the *MCP IP Address* field, the Web Services software on the host will use the network adapter that corresponds to that IP address for all outbound connections for all transactions done to the operations on the selected port. If this field is blank, the MCP will select the network adapter to be used. WSUtility does not check that the IP address you enter in this field is valid for the host system.

By default, the WebOut library on the MCP system includes the "Connection: close" HTTP header in each request, and closes the TCP/IP connection after the response is received. If the web service server (that provides the web service functionality) supports persistent connections, you can check the *Keep Alive* option to greatly enhance transaction throughput. If this option is checked, the WebOut library includes the "Connection: Keep-Alive" HTTP header in each request, and it does not close the port file when the transaction completes.

If you check the *Force MCP applications to provide a server URL* option, calling MCP applications will be forced to provide a server URL (the default server URL from the WSDL will be ignored).

If you select a port in the *Navigation Tree*, the default server URL for the port will be displayed in the *Port Server URL* field. This field is disabled for WSDL-based web services (you can not modify the URL).

## 9.2.3 Disabling Web Service Operations

By default, all operations defined in the WSDL file will be included in the host configuration file uploaded to the host, and will be callable by host programs.

To mark an operation as disabled (so it will not be included in the host configuration file), select it in the *Navigation Tree*, then click **Edit | Disable operation**, or right-click on the operation and click the **Disable operation** item on the popup menu. Disabled operations will be listed in the *Navigation Tree* with a ~~strikethru~~ font. You can re-enable an operation on the **Edit** menu or popup menu as well.

You can also disable or enable all operations on a port by selecting the port in the *Navigation Tree*, and then clicking the **Edit | Disable all operations** or **Edit | Enable all operations** menu items, or by clicking the same menu items from the popup menu. If you select the Web Service name in the *Navigation Tree*, you can enable or disable all operations on all ports at once.

### 9.2.4 Request and Response Records

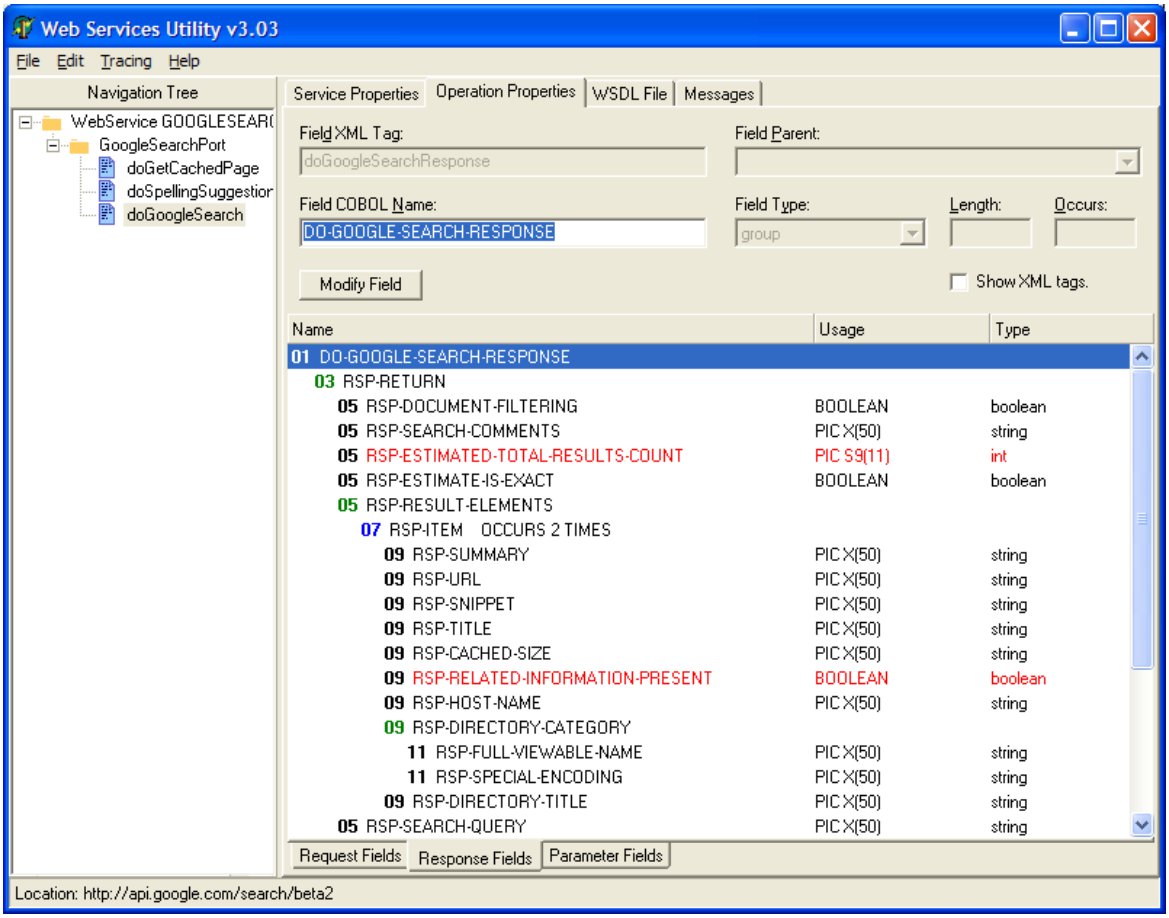
Once the WSDL file has been read in, WSUtility will convert the request and response message formats defined in the WSDL to flat record descriptions suitable for use by COBOL programs. The default data conversions are shown below:

| WSDL Type | COBOL Type |
|---|---|
| string | PIC X(50) |
| integer | PIC S9(11) |
| boolean | PIC X ("Y" or "N"; displayed in WSUtility as "BOOLEAN") |
| decimal, float, double | REAL |
| array | OCCURS 2 TIMES |
| <anything else> | PIC X(50) |

If the WSDL contains field length and array size information, WSUtility will use that information to set the field's length and OCCURS values. Otherwise, the default values will be used.

You can change the length of the "PIC X" string fields, the length of the "PIC S9" numeric fields, and the value of the OCCURS clause on any array. Alphanumeric (PIC X) fields are limited to 65,535 characters, numeric fields (PIC S9) are limited to 23 digits, and entire records are limited to one megabyte (1,048,575 characters).

To edit an operation's record descriptions, click the operation's name in the *Navigation Tree* on the left side of the WSUtility window. The *Operation Propertiess* tab will be automatically selected. As you can see in the image below, the record display is very similar to a COBOL "01" record description.




**Operation Response Fields**

To modify a field, select it with the mouse – its name, type, length, and occurs values will then be displayed in the fields above the record display. Any field data that you can not change (such as the XML tag for the field) will be grayed out and disabled. After you have changed one or more of the editable values, click the **Modify** button to apply the change.

***Note:*** WSUtility adds a REQ- prefix to each field in the request record, and a RSP- prefix to each field in the response record. This is to ensure that no field names are COBOL reserved words.

The level numbers have different colors, based on the field type – group fields that are not arrays are displayed in green; group fields with an OCCURS clause are blue, and elementary fields are black.

The field names displayed in red are too long; COBOL is limited to 30 character variable names. You must manually change these names before generating the host COBOL "COPY" file.

You can check the *Show XML tags* option to see the XML tag for each field as defined in the WSDL file.

Use the tabs at the bottom of the *Operation Fields* page to switch between the request, response and parameter records.

### 9.2.5 Parameter Records

WSUtility generates a "parameter record" in the COBOL include file for all operations on every port. The operation parameter record is used by your application to specify the operation it wishes to call. The COBOL name of each parameter record defaults to "<operation name>-PARAM", but you can modify the name to any valid COBOL identifier.

See the *Parameter Record Fields* section later in this document for details on the fields in the operation parameter records.

### 9.2.6 WS-Out SSL Operation

The use of SSL in WS-Out web services transactions is controlled by the server URL used when calling the web service (either from the WSDL, or as specified by the application in the parameter record when it calls the web service). If the URL specifies the Secure Socket Layer protocol (that is, it starts with "https://"), then SSL will be used for the outbound web services call. Otherwise, SSL will not be used, and the normal non-secure HTTP protocol is used for the web service's transactions.

### 9.2.7 Complex XML Schemas

WSUtility supports most commonly used XML Schema types and options when importing a WSDL. The following notes describe the known limitations and other special items of interest.

- The "mixed" schema option is not supported.
- The "choice" schema option is not fully supported; WSUtility will use the first specified choice, and ignore the other choices.
- The "anyAttribute" element is supported; WSUtility will generate a special field for the application to provide attributes and their values. Note that the application must ensure that the data inserted into this field is valid attribute syntax (ie, *attribute="value"*), as it is not validated by MGSWeb.
- The "any" element, and elements with a type of "anyType", are also supported in the request and response records. For these "anyType" fields, the application can put raw XML in the field in the request record, and MGSWeb will return raw XML in the field in the response record. Note that it is up to the application to ensure that the XML it places into request fields is valid XML, as MGSWeb does not validate the contents of the field.

- Only the “Request/Response” transmission primitive is supported; “Response/Request,” “Request Only” and “Response Only” are not supported.

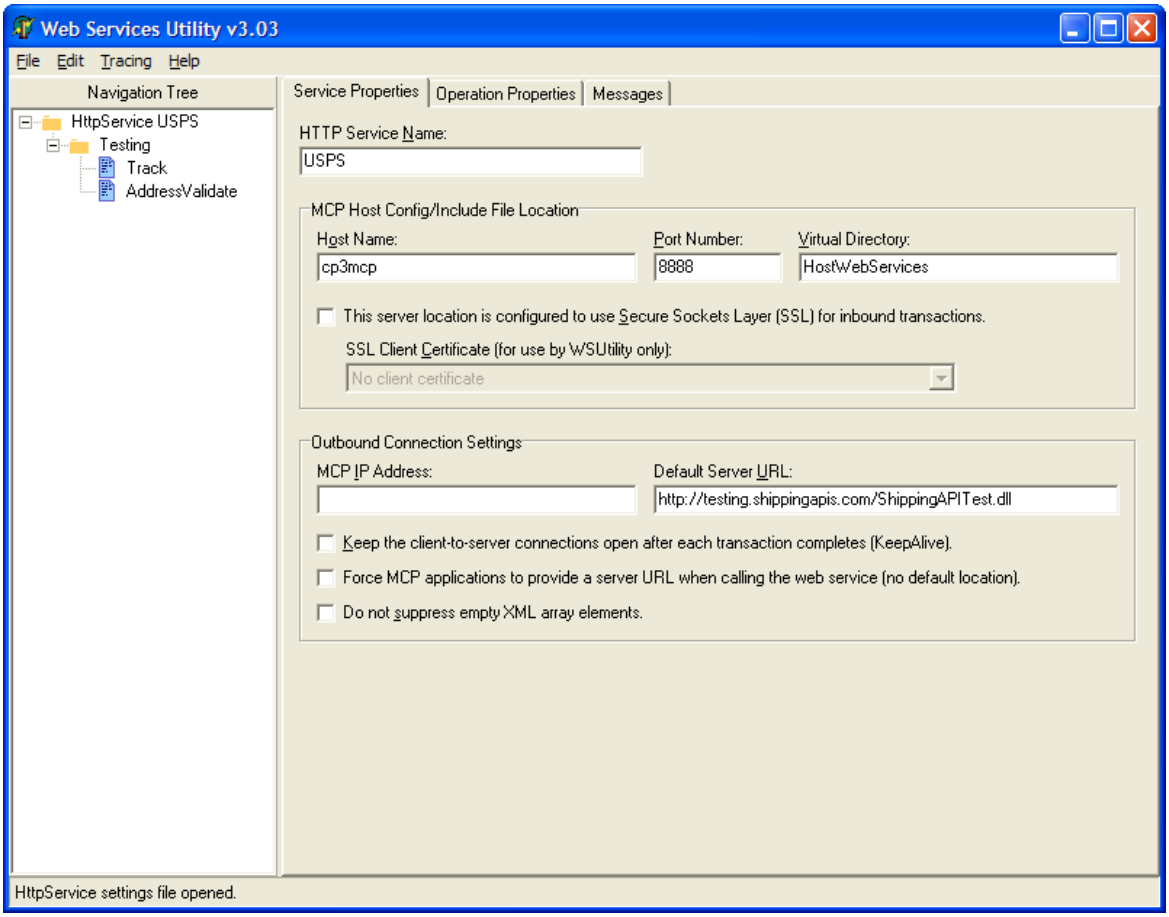
- The “minOccurs” schema option is completely supported. WS-Out only includes empty elements if the “minOccurs” value for the element is greater than zero. For arrays of elements, WS-Out includes at least the “minOccurs” number of elements, and up the OCCURS value set in WSUtility, if the extra occurrences are non-blank.

- If numeric fields (integer or floating point) in the request record contain all spaces, they are not included in the generated XML, unless they are defined in the WSDL with a “minOccurs” value greater than zero. If a numeric field contains all spaces and has a “minOccurs” value greater than zero, it will be inserted in the XML as *<tag/>*. If a numeric field has a zero value (but is not all spaces), it will be included in the XML as *<tag>0</tag>*. One oddity of this behavior is that if a REAL field in the request record contains the bit pattern 4”404040404040” (value -275955859520.0), it will not be included in the XML because it is all spaces.

## 9.3 Non-WSDL-Based Web Services

To create a new non-WSDL-based web service ("HTTP Service") project, click the **File | New | HTTP Service project** menu item. To open an existing HTP Service project, click the **File | Open | HTTP Services project** menu item. HTTP Service project files have a ".hsf" file extension.

After you have opened or created a new HTTP Service project, the WSUtility window will change to look like this:




**WSUtility with a new HTTP Service Project**

### 9.3.1 Service Properties

Enter the name of the HTTP Service in the *HTTP Service Name* field on the "Service Properties" page. HTTP Service names are limited to 17 uppercase characters.

**MCP Host Config/Include File Location**

You have to enter the DNS name or IP address, port number, virtual directory and SSL option (and optionally a client certificate name) for the ATLAS or Web Services server on your MCP system in the *MCP Host Config/Include File Location* fields. These are needed so the WSUtility program can transfer the host configuration and COBOL "COPY" files to the host.

**Outbound Connection Settings**

If you enter an IP address in the *MCP IP Address* field, the Web Services software on the host will use the network adapter that corresponds to that IP address for all outbound connections for all transactions done to the operations on the selected port. If this field is blank, the MCP will select the network adapter to be used. WSUtility does not check that the IP address you enter in this field is valid for the host system.

You use the *Default Server URL* field to specify the default server URL for the HTTP Service. Applications can override this value; see the *Parameter Record Fields* section later in this document for details.

By default, the WebOut library on the MCP system includes the "Connection: close" HTTP header in each request, and closes the TCP/IP connection after the response is received. If the web service server (that provides the web service functionality) supports persistent connections, you can check the *Keep Alive* option to greatly enhance transaction throughput. If this option is checked, the WebOut library includes the "Connection: Keep-Alive" HTTP header in each request, and it does not close the port file when the transaction completes.

If you check the *Force MCP applications to provide a server URL* option, calling MCP applications will be forced to provide a server URL (the value in the *Default Server URL* field will be ignored).

If you do not want the Web Services software to automatically suppress empty array elements, select the *Do not suppress empty XML array elements* option.

## 9.3.2 Operations

An operation is the name WSUtility uses to refer to a web service function to be called by your MCP-based program. When you define an operation in WSUtility, you must give it a name, and you must define the fields in both the request record and in the response record.

By default, when you create a new HTTP Service project, WSUtility creates a single port named *newPort_0* with a single operation named *newOperation_0*. To create additional operations, click the **Edit | Add operation** menu item, or right-click the HTTP Service or port in the *Navigation Tree*, and select the **Add operation** item from the popup menu.

To rename an existing operation, either select the operation in the *Navigation Tree* and click **Edit | Rename operation**, or right-click on the operation in the *Navigation Tree* and select the **Rename operation** item on the popup menu.

To delete an existing operation, either select the operation in the *Navigation Tree* and click the **Edit | Delete operation** menu item, or right-click on the operation in the *Navigation Tree* and select the **Delete operation** item from the popup menu. You can not delete the last operation on a port.

### 9.3.3 Request and Response Records

If you select an operation in the *Navigation Tree*, the *Operation Properties* page will be displayed so you can define the request and response fields, and view the parameter record fields, for the selected operation. You select which record you want to view using the tabs at the bottom of the page ("Request Fields," "Response Fields" or "Parameter Fields").

The *Operation XML Prefix* field allows you to specify optional text that will be inserted in all of the operation's requests, immediately preceding the XML itself.

**Defining a New Field**

To define a new field:

- Enter the XML tag for the field in the *XML Tag* field.
- Select the field's parent in the *Field Parent* list. A field's parent must be a "group" field; if you select a field in the *Field Parent* list that is not a "group" field, the new field will have the same parent as the selected field.
- Enter the field's COBOL name in the *COBOL Name* field, or leave this field blank to use the default COBOL name (which will be based on the field's XML tag).
- Select the type for the field (either "group", "string", "integer", "boolean", "double", or "attribute") using the *Field Type* list.
- Enter the length of the field in the *Length* field (except for "group" fields, which can not have a length).
- If the new field is an array, enter the length of the array in the *Occurs* field (except for "attribute" fields, which can only occur once).
- Click the **Add** button.

A "group" field can not have data associated with it; that is, it can not have a length. An "attribute" field equates to an XML attribute of its parent element; only "group" fields can have attributes.

The following table lists the field types (other than “group”) and their corresponding COBOL picture clauses:

| WSDL Type | COBOL Type |
|---|---|
| string | PIC X(<length>) |
| integer | PIC S9(<length>) |
| boolean | PIC X |
| float | REAL |
| attribute | PIC X(<length>) |

**Modifying and Deleting Existing Fields**

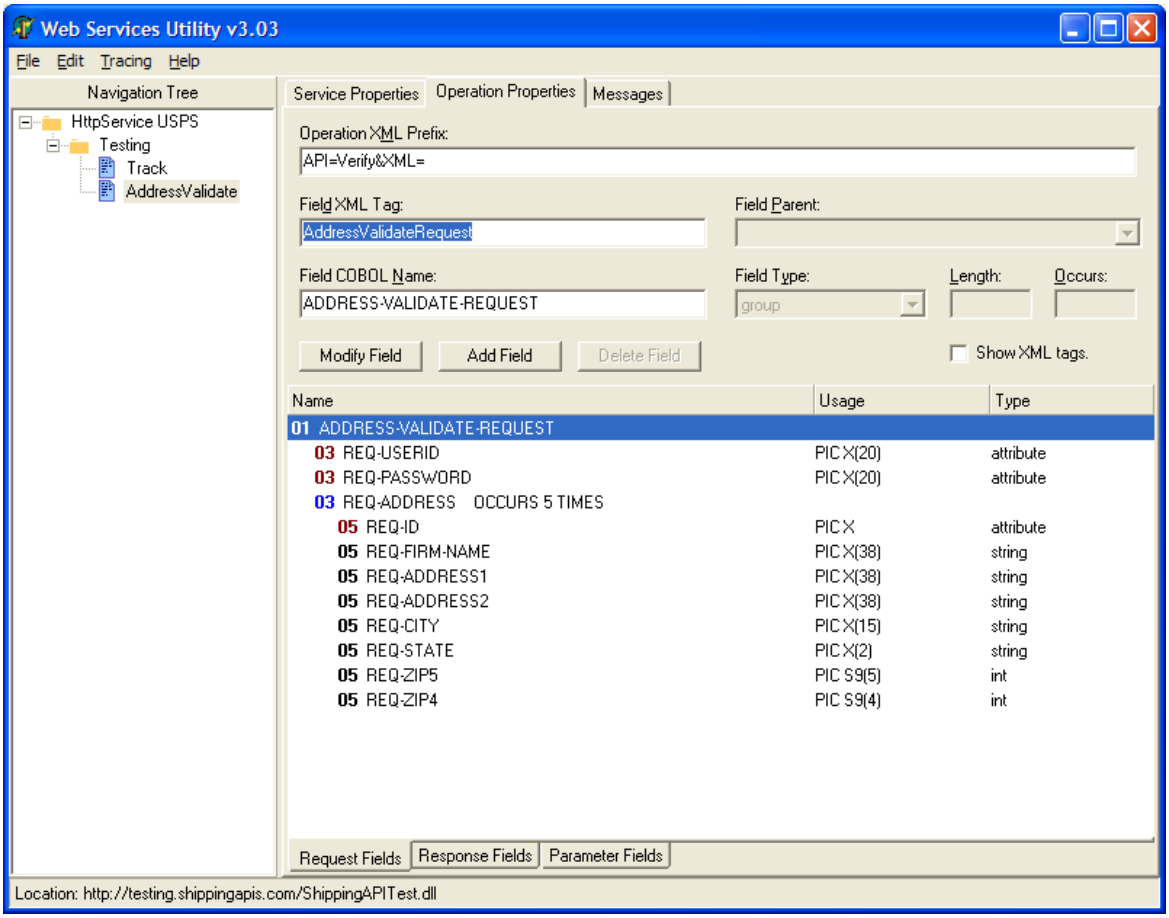
To modify a field, select it with the mouse, make the desired changes to the field’s values, and click the **Modify** button.

To delete a field, select it with the mouse, and click the **Delete** button.

You can also use “drag and drop” in the field list to reorder fields, or change a field’s parent, within the same record.

### Example Request Record

Here is an example request record for the US Postal Service's "AddressValidate" web service:




**The WSUtility with a USPS HTTP-Out Example**

The level numbers have different colors, based on the field type – group fields that are not arrays are displayed in green; array groups and fields with an OCCURS value greater than one are blue, attributes are maroon, and elementary fields are black.

***Note:*** Alphanumeric (PIC X) fields are limited to 65,535 characters, numeric fields (PIC S9) are limited to 23 digits, and entire records are limited to one megabyte (1,048,575 characters).

### 9.3.4 Parameter Records

WSUtility generates a “parameter record” in the COBOL include file for each operation on every port. The operation parameter record is used by your application to specify the operation it wishes to call. The COBOL name of each parameter record defaults to “<operation name>-PARAM”, but you can modify the name to any valid COBOL identifier.

See the *Parameter Record Fields* section later in this document for details on the fields in the operation parameter records.

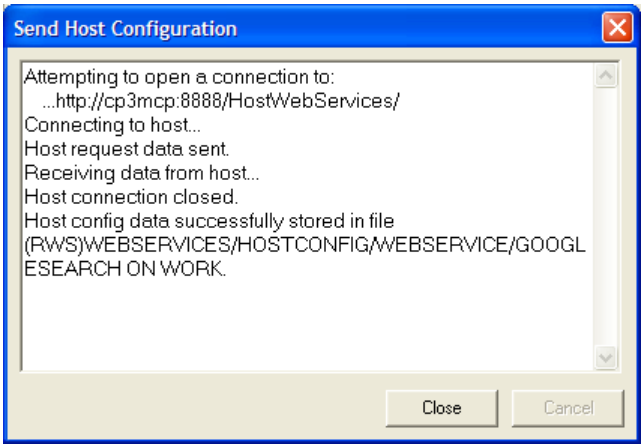
### 9.3.5 HTTP-Out SSL Operation

The use of SSL in HTTP-Out web services transactions is controlled by the server URL used when calling the web service (either from the *Default Server URL* field, or as specified by the application in the parameter record when it calls the web service). If the URL specifies the Secure Socket Layer protocol (that is, it starts with “https://”), then SSL will be used for the outbound web services call. Otherwise, SSL will not be used, and the normal non-secure HTTP protocol is used for the web service’s transactions.

## 9.4 Generating the Host Configuration File

After you have completed desired changes to the record descriptions, you can generate the host configuration file. The host configuration file describes the web service in detail, so the host software can encode the requests and decode the responses.

To generate the host configuration file, click the **File | Generate host config file** menu item. You must have the host Web Services software running before doing this, because WSUtility attempts to open a connection to the host Web Services software to transfer the configuration file. When the host configuration file has been successfully transferred, the name of the file created on the host will be displayed in the *Send Host Configuration* dialog, as shown below.




**WSUtility Host Configuration Update Dialog Box**

The host configuration files for both WSDL-based and non-WSDL-based web services are stored in the `WEBSERVICES/HOSTCONFIG/WEBSERVICES/=` directory under the same usercode and on the same family as the Web Services host software.

## 9.5 Generating the COBOL "COPY library"

If you will be using COBOL to access the web service, you can generate a COBOL "COPY library" (include file) that you can copy into your COBOL source to declare the parameter, request and response "01" record descriptions. The include file also will contain comment records listing the web service, port, and available operation names.

To generate the COBOL include file, click the **File | Generate host include file** menu item. As with the host configuration file, the Web Services host software must be running before you do this. When the include file has been successfully transferred, the name of the file created on the host will be displayed in the *Send Host Configuration* dialog, as shown previously. The include file will be uploaded to the same usercode and family as the Web Service code files, and will be named:

```
WEBSERVICES/USERFILES/WEBSERVICE/<web/http service name>
```

Here is an example COBOL include file for the GOOGLESEARCH web service shown earlier:

```
000100* WebService GOOGLESEARCH
000200* Port GoogleSearchPort
000300* Operation doGoogleSearch * * * * * * * * * * * * * * * * * * * *
000400 01 DO-GOOGLE-SEARCH-PARAM.
000500   03 WEBSERVICE-NAME                          PIC X(24) VALUE
000600        "GOOGLESEARCH".
000700   03 PORT-NAME                                PIC X(150) VALUE
000800        "GoogleSearchPort".
000900   03 OPERATION-NAME                           PIC X(150) VALUE
001000        "doGoogleSearch".
001100   03 OPERATION-SIGNATURE                      REAL VALUE
001200        18946402.
001300   03 NETWORK-TIME                             REAL.
001400   03 TRANSACTION-TIMEOUT                      REAL.
001500   03 TRANSACTION-TRACE                        REAL.
001600   03 SERVER-LOCATION                          PIC X(150).
001700   03 SERVER-USERNAME                          PIC X(60).
001800   03 SERVER-PASSWORD                          PIC X(60).
001900   03 KEY-NAME                                 PIC X(60).
002000   03 FILLER                                   PIC X(522).
002100 01 DO-GOOGLE-SEARCH.
002200   03 REQ-KEY                                  PIC X(50).
002300   03 REQ-Q                                    PIC X(50).
002400   03 REQ-START                                PIC S9(11).
002500   03 REQ-MAX-RESULTS                          PIC S9(11).
002600   03 REQ-FILTER                               PIC X.
002700   03 REQ-RESTRICT                             PIC X(50).
002800   03 REQ-SAFE-SEARCH                          PIC X.
002900   03 REQ-LR                                   PIC X(50).
003000   03 REQ-IE                                   PIC X(50).
003100   03 REQ-OE                                   PIC X(50).
003200 01 DO-GOOGLE-SEARCH-RESPONSE.
003300   03 RSP-RETURN.
003400     05 RSP-DOCUMENT-FILTERING                 PIC X.
003500     05 RSP-SEARCH-COMMENTS                    PIC X(50).
003600     05 RSP-EST-TOTAL-RESULTS-COUNT            PIC S9(11).
003700     05 RSP-ESTIMATE-IS-EXACT                  PIC X.
003800     05 RSP-RESULT-ELEMENTS.
003900       07 RSP-ITEM OCCURS 2 TIMES.
004000         09 RSP-SUMMARY                        PIC X(50).
004100         09 RSP-URL                            PIC X(50).
004200         09 RSP-SNIPPET                        PIC X(50).
004300         09 RSP-TITLE                          PIC X(50).
004400         09 RSP-CACHED-SIZE                    PIC X(50).
004500         09 RSP-RELATED-INFO-PRESENT           PIC X.
004600         09 RSP-HOST-NAME                      PIC X(50).
004700         09 RSP-DIRECTORY-CATEGORY.
004800           11 RSP-FULL-VIEWABLE-NAME           PIC X(50).
004900           11 RSP-SPECIAL-ENCODING             PIC X(50).
005000         09 RSP-DIRECTORY-TITLE                PIC X(50).
005100     05 RSP-SEARCH-QUERY                       PIC X(50).
005200     05 RSP-START-INDEX                        PIC S9(11).
005300     05 RSP-END-INDEX                          PIC S9(11).
005400     05 RSP-SEARCH-TIPS                        PIC X(50).
005500     05 RSP-DIRECTORY-CATEGORIES.
005600       07 RSP-ITEM OCCURS 2 TIMES.
005700         09 RSP-FULL-VIEWABLE-NAME             PIC X(50).
005800         09 RSP-SPECIAL-ENCODING               PIC X(50).
005900     05 RSP-SEARCH-TIME                        REAL.
```

WSUtility does not currently support generating ALGOL include files, so you will have to manually create a list of EBCDIC array DEFINEs for the request, response and parameter records if you want to call the web service from an ALGOL program.

# 10. Using WS-Out or HTTP-Out to Call a Web Service

After generating the host configuration file and the COBOL include files, you can add code to your COBOL or ALGOL program to call the web service.

To call a web service, your application must call the `INVOKE` procedure in the `WEBSERVICES/LIBRARY` library. The `INVOKE` procedure accepts four parameters, and returns an `INTEGER` result. The return value of the `INVOKE` procedure will be zero if the call completed successfully, and non-zero if an error occurs.

## 10.1 COBOL Syntax

For COBOL, the code to call the INVOKE procedure call is:

```
01 RESULT-STRING            PIC X(256).
.
77 RESULT                   PIC 9(11) BINARY.
.
<move data to the request record fields>
.
CALL "INVOKE OF WEBSERVICES/LIBRARY"
  USING <parameter record>,
        <request record>,
        <response record>,
        RESULT-STRING
  GIVING RESULT.
```

## 10.2 ALGOL Syntax

The ALGOL definition for the `INVOKE` procedure is:

```
LIBRARY WEBSERVICES (TITLE="WEBSERVICES/LIBRARY.");

INTEGER PROCEDURE INVOKE
      (PARAMETER_RECORD, REQUEST_RECORD,
       RESPONSE_RECORD, RESULT_STRING);
EBCDIC ARRAY PARAMETER_RECORD, REQUEST_RECORD,
             RESPONSE_RECORD, RESULT_STRING [0];
  LIBRARY WEBSERVICES;
```

## 10.3 INVOKE Procedure Parameters

The first parameter to the INVOKE function is the parameter record for the operation you want to call. For COBOL applications, the operation parameter records are defined in the COBOL include file generated by WSUtility; for an ALGOL application, you will have to manually declare a 1,200 byte EBCDIC array for this parameter. The parameter record contains the web service, port and operation names for the operation you want to execute, as well as other control fields. See the next section for details on the parameter record's fields.

The second and third parameters to INVOKE are the request and response buffers. For COBOL applications, these two parameters are defined in the COBOL include file generated by WSUtility; for ALGOL applications, you will have to declare them manually in your program. You place the parameter values for the web service call into the request buffer, and the response values from the call will be placed in the response buffer.

The last parameter, RESULT-STRING, will contain an error message, ending with a null, if the web service call fails (that is, the return value of the call is not zero). It will be resized, if necessary, to fit the entire message. If no error occurs, the text "No error" will be placed into this parameter. You must declare this array in your application.

If you set the TRANSACTION-TRACE field of the parameter record to a non-zero value, the Web Service software will enable host tracing for this single transaction. The request and response XML, and the COBOL record field values, will be written to the WEBSERVICES/LOG file.

## 10.4 Parameter Record Fields

WSUtility includes a "parameter record" for each enabled operation in the COBOL include files it generates. The operation parameter records are used by your applications to call a specific operation. The operation parameter record must be 1,200 bytes (200 words) long.

Each operation parameter record contains the following fields:

WEBSERVICE-NAME (24 characters)
PORT-NAME (150 characters)
OPERATION-NAME (150 characters)

> These three fields are used by the web services software to select the operation to be executed. For COBOL applications, these fields are set to the correct values in the COBOL include file; you do not have to set them. For ALGOL programs, you will have to put the correct values in the fields; see the COBOL include file for the correct values.

`OPERATION-SIGNATURE` (6-byte REAL)

This field contains a value that allows the web services software to verify that the request and response records for the operation your application is calling match the operation's request and response records in the current host configuration file. This allows the web services software to detect if the COBOL include file compiled into your application matches the current host configuration file for the operation your application is calling. For COBOL applications, this field is set to the correct value for you. For ALGOL applications, you will have to set this field to the correct value; see the COBOL include file for the correct signature value. If you set this field to zero, the signature checking will not be done.

`NETWORK-TIME` (6-byte REAL)

Upon exit from the `INVOKE` function, this field will contain the network elapsed time, in milliseconds, for the completed transaction when the operation completes. Network elapsed time consists of all TCP/IP socket time, including open/close and send/receive.

`TRANSACTION-TIMEOUT` (6-byte REAL)

The normal transaction timeout is about three minutes; if you want a shorter timeout value, you can specify it in this field (in seconds). Set this field to zero to use the default timeout.

`TRANSACTION-TRACE` (6-byte REAL)

If your application sets this field to a non-zero value, the raw request and response XML, and the COBOL record field values, will be written to the `WEBSERVICES/LOG` file for just this single transaction.

`SERVER-LOCATION` (150 characters)

By default, the Web Services software uses the destination server URL specified in the WSDL (for WS-Out services) or the *Default Server URL* field of WSUtility's *Service Properties* page (for HTTP-Out services). To use a different server URL, your application can specify it in this field. The general format of a valid URL is **protocol://hostname:portnumber/directory**. The protocol must be either HTTP or SSL; that is, the URL must start with either "http://" or "https://". If the URL does not supply a port number, port 80 will be used for HTTP and port 443 will be used for SSL. If the *Force MCP applications to provide a server URL* option is enabled, the calling application **must** provide a URL in this field, or an error will be returned by the `INVOKE` procedure.

SERVER-USERNAME (60 characters)
SERVER-PASSWORD (60 characters)

If the web server requires authentication, you must specify a user name and a clear text password in these fields so WS-Out or HTTP-Out can authenticate the connection with the server. MGSWeb supports the following authentication schemes: Windows NTLM, HTTP Basic and HTTP Digest. MGSWeb uses the "WWW-Authenticate" HTTP headers returned by the server to automatically select the authentication scheme to use; if the server supports multiple schemes, MGSWeb will select a scheme in the following order: Basic first, then NTLM, and lastly Digest.

While the application does require prior knowledge that a username and password will be needed for a given WS-Out or HTTP-Out call, knowledge of specifically which authentication scheme to use is not the application's responsibility.

Note that the Windows NTLM and HTTP Digest authentication schemes are only supported if the MGSWeb Authentication Extensions are enabled in the MGSWeb host software AccessKey.

KEY-NAME (60 characters)

If the remote server is configured to use two-way SSL, the calling application can specify the name of the "key container" containing the client certificate to be used when connecting to the server.

HTTP-HEADERS (150 characters)

Use this field to specify zero or more custom HTTP headers to be added to the outbound request. The format of each header in this field should be:

```
<header name><colon><double quote><value><double quote><space>
```

The <double quote> characters are optional if the <value> does not contain any embedded spaces. The following headers are ignored if specified in this field: "Content-Type", "Content-Length", "Transfer-Encoding", "Authorization", "Connection" and "SOAPAction".

## 10.5 INVOKE Result Values

The return value of the INVOKE procedure will be zero if the call completed successfully, and non-zero if an error occurs. The following return values are currently defined:

NO_ERRORV (0)

The web service call completed successfully.

`INITIALIZATION_ERRORV` (1)

Either the web service name in the parameter record is not valid, or the host configuration file for the specified web service is missing or invalid.

`PORT_ERRORV` (2)

The port name specified in the parameter record is not valid.

`OPERATION_ERRORV` (3)

The operation name specified in the parameter record is not valid.

`COMMUNICATION_ERRORV` (4)

A TCP/IP communication error occurred during the web service call. This can be caused by the server being offline, or the web service being down or moved to a different URL.

`HTTP_ERRORV` (5)

An unexpected HTTP status code (other than 200) was received from the Web Services server. The `RESULT-STRING` parameter will contain further details.

`XML_FORMAT_ERRORV` (6)

The response from the Web Services server was not valid XML.

`XML_PROCESSING_ERRORV` (7)

The XML parser generated an error; the `RESULT-STRING` parameter will contain the XML parser's error.

`SOAP_FAULT_ERRORV` (8)

The web service response contained a SOAP fault; the `RESULT-STRING` parameter will contain the SOAP fault code and SOAP fault string.

`RESPONSE_ERRORV` (9)

An unexpected response was received; that is, the parameters of the response did not match the definition in the host configuration file.

`WEB_SERVICES_DISABLEDV` (10)

Outbound Web Services calls are not enabled in your MGSWEB access key.

`HTTP_SERVICES_DISABLEDV` (11)

Outbound HTTP services calls are not enabled in your MGSWEB access key.

`REQUEST_BUFFER_ERRORV` (12)

The request buffer is too short. For COBOL applications, this error is the result of passing in the wrong request buffer to the `INVOKE` function.

`RESPONSE_BUFFER_ERRORV` (13)

The response buffer is too short. For COBOL applications, this error is the result of passing in the wrong response buffer to the `INVOKE` function.

`SIGNATURE_ERRORV` (14)

The signature value in the operation parameter array does not match the signature value from the host configuration file for the operation. This error is caused by a mismatch between the copy library included into the calling application and the current host configuration file.

The most common reason for this is that the request or response records have been changed and a new host configuration file re-generated, but either the copy library was not also re-generated or the application was not re-compiled using the new copy library.

`PARAM_ARRAY_ERRORV` (15)

The operation parameter array was too short (it must be 1,200 bytes long).

`WORKER_ABORT_ERRORV` (16)

The WEBSERVICES/WEBOUT worker task aborted while executing the application's transaction. This error can only be returned if the application sets the `TRANSACTION-TIMEOUT` field in the parameter record to a non-zero value.

`TIMEOUT_ERRORV` (17)

The outbound web services transaction timed out and was aborted. This error can only be returned if the application sets the `TRANSACTION-TIMEOUT` field in the parameter record to a non-zero value.

`URL_ERRORV` (18)

The URL specified in the SERVER-LOCATION field in the parameter record is not valid; either the protocol is not HTTP or SSL, or the port number is out of range.

## 10.6 Host Tracing

The easiest way to enable host tracing for your WS-Out or HTTP-Out transactions is to set the TRANSACTION-TRACE field in the parameter record to a non-zero value. This will cause MGSWeb to write trace information to the WEBSERVICES/LOG file for the single transaction.

You can also enable host tracing for a specific project using WSUtility's **Tracing** menu, the TRACE operator command to the DRIVER program, or by adding the TRACE command to the WEBSERVICES/PARAM file.

When the host trace option is enabled for a project, all transactions to that Station, Program, Database or WebService project cause the request and response XML, and the contents of the COBOL input and output records, to be written to the WEBSERVICES/LOG file. All message traffic is included in the tracing information, including the HTTP headers for WS-Out and HTTP-Out requests.

# 11. Web Services for Unisys MCP Sample Programs

This section details how to set up and run the WS-In and WS-Out samples included with the Web Services for Unisys MCP release.

Before running any of these sample programs, you must have already installed and started the Web Services for Unisys MCP host software. For detailed installation instructions, see section 2 of this document.

## 11.1 WS-In Sample Program

The WS-In sample consists of a COMS direct window program, called WEBINSAMPLE, that converts a 50 character field from lower case to upper case characters and outputs the Unisys MCP system's host name. The sample program can handle input from either a station or from the Web Services RESPONDER program.

The following sections contain detailed instructions for compiling and defining the sample COMS program on your MCP system.

### 11.1.1 Compiling the Sample COMS Program

The following steps are necessary to compile the WS-In sample COMS program:

1. Log on to CANDE under the usercode you installed the host software under (or a privileged usercode if you installed the Web Services software un-usercoded).

2. To compile the WS-In sample COMS program with the DMALGOL compiler, enter the following command:

   ```
   COMPILE WEBSERVICES/SAMPLES/WEBIN/UPPERCASE
   ```

3. If you want to use the ALGOL compiler instead, enter the command:

   ```
   COMPILE WEBSERVICES/SAMPLES/WEBIN/UPPERCASE WITH ALGOL
   ```

The compiled program will be named:

```
OBJECT/WEBSERVICES/SAMPLES/WEBIN/UPPERCASE
```

### 11.1.2 Configuring the Sample COMS Program

The WS-In sample COMS load deck contains the commands necessary to create a COMS program, window and agenda, all named WEBINSAMPLE, in the COMS CFile. To load the WS-In sample COMS load deck:

1. Transfer to the COMS Utility window with the command **?ON UTILITY**.

2. Go to the Load screen by entering the command **GO LO** in the *Action* field.

3. On the Load screen, enter the **LO** command in the *Action* field and **(<usercode>)WEBSERVICES/SAMPLES/WEBIN/COMSLOADDECK ON <family>** in the *Load File Title* field (where **<usercode>** is the usercode you loaded the Web Services software under, and **<family>** is the family you loaded the software to), then transmit.

4. You should get the message "Load Command was successfully completed" at the bottom of the screen.

If you are running the Web Services software under a usercode, or you loaded the Web Services software un-usercoded to a family other than DISK, you will need to modify the program definition for the WS-In sample COMS program. To do this:

1. Go to the Utility Program screen with the command **GO P** in the *Action* field.

2. On the Program screen, enter the **INquire** command in the *Action* field, **WEBINSAMPLE** in the *Program Name* field, and press transmit.

3. Enter the **MOdify** command in the *Action* field.

4. If you are running the Web Services software under a usercode, enter that usercode in the *USERCODE* field. If you did not load the Web Services software to the usercode's primary family, add **ON <family>** (where **<family>** is the family you loaded the Web Services software to) to the end of the program title in the *TITLE* field.

5. If you are running the Web Services software un-usercoded, but you did not load the software to family DISK, add **ON <family>** (where **<family>** is the family you loaded the Web Services software to) to the end of the program title in the *TITLE* field.

6. Press transmit to store your changes.

Close the Utility window with the command **?CLOSE**.

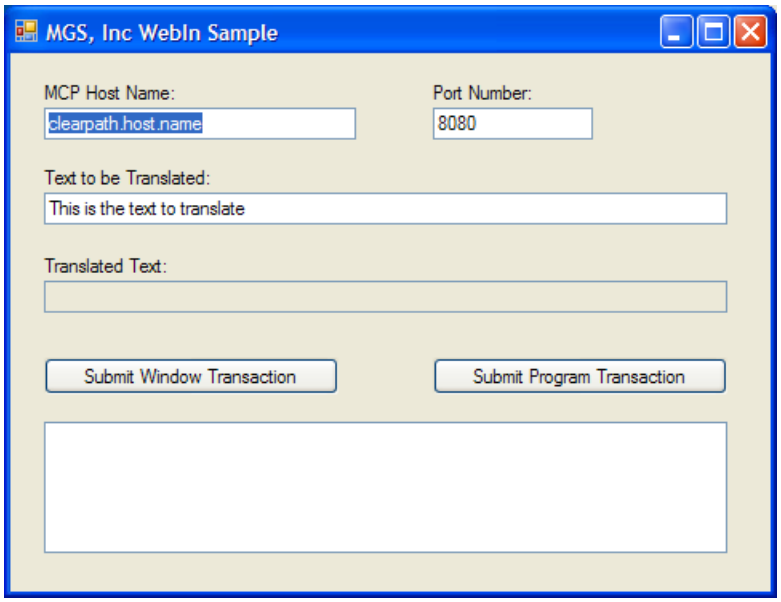
### 11.1.3 Running the WEBINSAMPLE Visual Basic Program

After you have compiled and defined the WEBINSAMPLE COMS program on the Unisys MCP system, you can run the corresponding Visual Basic .NET program, also called WEBINSAMPLE, on your PC.

The WS-In sample program, called `WebInSample.exe`, is located in the `Samples\VB Project\` directory. It allows you to call the WEBINSAMPLE web service on the Unisys MCP system using either a Station transaction or a Program transaction.

***Note:*** In order for Program transactions to work, you must have defined and enabled the Web Services RESPONDER program on the Unisys MCP system.

The WEBINSAMPLE window is shown below:




**Web-In Sample Program Window**

Before attempting to do any transactions, make sure you enter the TCP/IP host name of the Unisys MCP system you are running the Web Services software on in the *MCP Host Name* field. If you changed the port number from the default value (8080) in the `WEBSERVICES/SERVER/PARAM` file, or you are using ATLAS as your web server, you must also change the port number in the *Port Number* field.

Clicking the *Submit Station Transaction* button submits a Station transaction to the WEBINSAMPLE WS-In web service. Clicking the *Submit Program Transaction* button submits a Program transaction to the WEBINSAMPLE WS-In web service.

Status messages are added to the list box at the bottom of the window.

### 11.1.4 Deleting the COMS Sample Program

To delete the sample program from the COMS CFile:

1. Disable the WEBINSAMPLE program with the command **?DISABLE PROGRAM WEBINSAMPLE**.

2. Disable the WEBINSAMPLE window with the command **?DISABLE WINDOW WEBINSAMPLE**.

3. Transfer to the COMS Utility window with the command **?ON UTILITY**.

4. Go to the Load screen by entering the command **GO LO** in the *Action* field.

5. On the Load screen, enter the **LO** command in the *Action* field and **(<usercode>)WEBSERVICES/SAMPLES/WEBIN/COMSUNLOADDECK ON <family>** in the *Load File Title* field (where **<usercode>** is the usercode you loaded the Web Services software under, and **<family>** is the family you loaded the software to), then transmit.

6. You should get the message "Load Command was successfully completed" at the bottom of the screen.

## 11.2 WS-Out Sample Program

The WS-Out sample consists of a COBOL74 program on the Unisys MCP system, called `WEBSERVICES/SAMPLES/WEBOUT/CURRENCYCONVERTOR`, that calls a currency conversion web service on the www.webservicex.net web site.

You will need to compile this program on your MCP system before you can run it.

### 11.2.1 Compiling the Sample Program

To compile the WS-Out sample program, log on to CANDE under the usercode you installed the host software under (or a privileged usercode if you installed the Web Services software un-usercoded). Use the following command to compile the COBOL74 program:

```
COMPILE WEBSERVICES/SAMPLES/WEBOUT/CURRENCYCONVERTOR
```

The compiled program will be named:

```
OBJECT/WEBSERVICES/SAMPLES/WEBOUT/CURRENCYCONVERTOR
```

### 11.2.2 Running the Sample Program

The WS-Out sample program will call the INVOKE procedure in the `WEBSERVICES/LIBRARY`, which will load the host configuration file for the currency conversion web service (which is called `WEBSERVICES/HOSTCONFIG/WEBSERVICE/CURRENCYCONVERTOR`) and then call the web service on www.webservicex.net to convert US dollars to Euros.

To run the WS-Out sample program, log on to CANDE under the usercode you installed the host software under (or a privileged usercode if you installed the Web Services software un-usercoded).

To make sure DISPLAYs will be shown on your terminal, enter the command:

```
SO MESSAGE
```

To run the program, enter the command:

```
RUN WEBSERVICES/SAMPLES/WEBOUT/CURRENCYCONVERTOR
```

The result of the web service call will be DISPLAYed on your terminal. For example:

```
R WEBSERVICES/SAMPLES/WEBOUT/CURRENCYCONVERTOR
#RUNNING 5096
#5096 MGSWEB_WEBOUT:Loading (MGS)WEBSERVICES/HOSTCONFIG/WEBSERVICE/
     CURRENCYCONVERTOR ON WORK.
#5098\5099 BOT  (MGS)WEBSERVICES/WEBOUT/WORKER
#5096 DISPLAY:One US dollar is worth 0.7302 Euros (network time was 18441ms).
#ET=18.9 PT=0.1 IO=0.0
```

## 11.3 Included Sample Files

The WSUtility Program project for the WS-In sample program, called `WEBINSAMPLE.psf`, is included in the `Samples\` directory.

You can view the SOMS project for the WS-In sample Station transactions by opening the SOMS project in the `Samples\SOMS Project\WebInSample\` directory. You can also open this SOMS project using WSUtility to see the Station transaction settings.

If you want to view the source code for the WS-In sample program, the Visual Studio .NET project is located in the `Samples\VB Project\WebInSample\` directory.

The Web Services project file for the WS-Out sample, called `CURRENCYCONVERTOR.wsf`, is located in the `Samples\` directory. You can open this project file with WSUtility.

**Unisys MCP Sample Files**

The following Unisys MCP files are included with the Web Services for Unisys MCP software:

**`WEBSERVICES/SAMPLES/WEBIN/UPPERCASE`**

This is the WS-In sample COMS program; it supports transaction from both stations (Station transactions) and the RESPONDER program (Program transactions).

**WEBSERVICES/SAMPLES/WEBIN/COMSLOADDECK**

This file contains the commands necessary to define the COMS program, window and agenda for the WS-In sample COMS program.

**WEBSERVICES/SAMPLES/WEBIN/COMSUNLOADDECK**

This file contains the commands necessary to delete the COMS program, window and agenda for the WS-In sample COMS program from the COMS CFile.

**WEBSERVICES/HOSTCONFIG/WINDOW/WEBINSAMPLE**

This is the host configuration file that supports Station transactions for the WS-In sample web service.

**WEBSERVICES/HOSTCONFIG/PROGRAM/WEBINSAMPLE**

This is the host configuration file that supports Program transactions for the WS-In sample web service.

**WEBSERVICES/SAMPLES/WEBOUT/CURRENCYCONVERTOR**

This is the WS-Out sample COBOL74 program that calls a currency conversion web service on the www.webservicex.net web site.

**WEBSERVICES/HOSTCONFIG/WEBSERVICE/CURRENCYCONVERTOR**

This is the host configuration file that supports calling the currency conversion web service.

**WEBSERVICES/USERFILES/WEBSERVICE/CURRENCYCONVERTOR**

This is the COBOL copy library, generated by the WSUtility program, that defines the request, response and parameter records for calling the currency conversion web service. It is used by the WS-Out sample COBOL program.

### Windows PC Sample Files

The following PC sample files and directories are included with the Web Services for Unisys MCP release (in the Samples\ directory):

**WindowWEBINSAMPLE.wsdl**

This is the WSDL file for Station transactions, generated by WSUtility, for the WS-In sample web service.

**ProgramWEBINSAMPLE.wsdl**

This is the WSDL file for Program transactions, generated by WSUtility, for the WS-In sample web service.

**WEBINSAMPLE.psf**

This is the “program settings file” (psf) for the WSUtility Program project for the WS-In sample web service. You can open this project file in WSUtility.

**SOMS Project**

This directory contains is the SOMS project files for the WS-In sample Station transactions. (SOMS is used to define the fields on Unisys MCP application screens.) You can open this project in either SOMS or in WSUtility.

**VB Project\WebInSample.exe**

This is the sample Visual Basic .NET program that calls the WS-In sample COMS program using either Station or Program transactions.

**VB Project\WebInSample**

This directory contains the Visual Studio 2005 project for the sample Visual Basic .NET WS-In program. You can open the entire project using Microsoft Visual Studio, or use any text editor (such as NotePad) to open the file `VP Project\WebInSample\WebInSample\Form1.vb` to view the VB source for calling the web service.

**CURRENCYCONVERTOR.wsf**

This is the "web service settings file" (wsf) for the WSUtility Web Service project for the WS-Out sample web service. You can open this project file in WSUtility.

# Appendix I Release Notes

This section documents the technical changes made to the MGSWeb software since the 1.02 release.

## I.1 Release 1.03

The changes made as part of the 1.03 release are as follows:

1. The Web Services software will now automatically enable and disable the RESPONDER program when the DRIVER program initializes and terminates. A new parameter file command allows the user to control this feature ("`ENABLE_RESPONDER = ON | OFF;`"); the default setting for the new command is "`ON`".

2. The Web Services software now supports "temporary" access keys. Temporary keys work on any MCP system for a limited amount of time (an expiration date is required in a temporary key).

3. The major individual Web Services features (WS-In, WS-Out and HTTP-Out) can now be disabled in the access key.

4. The Web Services parameter file scanner now properly handles lower case characters in the parameter file.

5. The Web Services PSH library now terminates more quickly after the rest of the software has terminated (the PSH library was waiting up to five minutes on previous releases; it now will wait no more than 30 seconds before terminating).

6. The WEBIN sample COMSLOADDECK file has been enhanced to include a 30 second time-out for the sample program, so the program will terminate 30 seconds after the last user input.

7. A new COMSUNLOADDECK file has been added to the WEBIN samples files to delete the COMS entities added by the COMSLOADDECK file.

8. When a new WS-Out or HTTP-Out host configuration file was generated and uploaded to the host, the DRIVER program would continuously link to, and immediately delink from, LIBRARY until the DRIVER program was terminated and restarted. This has been corrected.

9. The WSUtility program has been enhanced to prompt the user if there are unsaved changes to the current project when the user attempts to create a new project, open an existing project, or exit the program.

10. The WSUtility program will now validate the current project's configuration before generating the host configuration file or the host include file. This will prevent the utility from creating invalid host configuration and include files.

11. If a WS-Out or HTTP-Out transaction received an invalid HTTP response from the web services server, it would report an "operation timed out" error. This has been corrected.

12. All error messages generated by WSUtility are now displayed as message boxes, rather than just as text in the status bar at the bottom of the WSUtility window.

13. A new testing feature has been added to WSUtility for WS-In transactions. After a station, program or database web service has been defined, and the host configuration file has been generated, you can use the new "Test" page to submit a transaction to the host software and view the response. The Test page also contains an "Allocate" button to allocate a station or database on the host, so you can submit transaction to a web service that does not permit stateless transactions.

14. The WEBSERVICES/LIBRARY code file has been renamed to WEBSERVICES/WEBOUT, and a new WEBSERVICES/LIBRARY has been implemented. The new LIBRARY will allow WS-Out and HTTP-Out users to install new software releases without bringing their applications down (this feature is not completely implemented yet). Users should still use the WEBSERVICES/LIBRARY library in their applications; they should not link directly to the WEBOUT library.

15. WS-Out transactions now support both "rpc/encoded" and "document/literal" web services.

16. WS-Out and HTTP-Out transactions also now support elements of type "any" and type "schema" in response messages. These elements will contain raw XML data when received by the MCP application.

17. **WS-In's web services have been changed from the "rpc/encoded" binding style to the "document/literal" binding style.** This will require existing WS-In sites to do the following for existing web services:

1. Regenerate the WSDL and host configuration files.

2. Re-import the new WSDL into their development environment for each application.

3. Make minor changes to their source code (the request record was called "<operation>Type", and it is now called "<operation>Request"; the response record was called "<operation>ResponseType", and it is now called "<operation>Response").

4. Re-compile their applications.

18. The WSUtility ASP project files have been updated to generate “document/literal” format transactions.

19. The WS-In host software’s error messages that are sent to the client have been improved to better explain the exact problem with the request data.

20. A new item, “MCP-NETWORK_TIME”, will be automatically added to all WS-Out and HTTP-Out response records. This item will contain the network time, in milliseconds, for the transaction (network time includes all TCP/IP connection operations, including open/close and send/receive). **This change requires that sites regenerate all WS-Out and HTTP-Out host configuration and COBOL include files, and recompile their applications.**

21. The MGS Web Services server has been enhanced to support persistent connections for WS-In transactions (ATLAS already supports persistent connections). Note that the MGS Web Services server does not support transaction pipelining over persistent connections (the ATLAS server does support pipelining).

22. The “Keep-Alive” option has been added to WSUtility for WS-Out and HTP-Out projects, to support persistent connections for outbound transactions.

23. WS-Out and HTTP-Out transactions were not timing out the TCP/IP port open attempt after 60 seconds (if for some reason the port file could not be opened); this has been corrected.

24. XML references starting with “&#x” (hex literals) in field data could cause a DRIVER worker stack “INVALID INDEX @ (00068200[XML])” fault; this has been corrected.

25. Minor changes have been made to WS-In and WS-Out to make them more compatible with the *WS-I Basic Profile Version 1.1* specification.

## I.2 Release 2.00

The changes for the 2.00 release are as follows:

1. Secure Socket Layer (SSL) connections are now fully supported for both inbound and outbound web services transactions. WSUtility now supports connecting to the host via an SSL connection to transfer host configuration and COBOL include files, and for testing inbound web services. The WS-Out host software now supports outbound SSL connections to the web services server (to support SSL, the WEBSERVICES/WEBOUT library now uses sockets via the SOCKETSUPPORT library, rather than TCPIPNATIVESERVICE port files, for outbound connections). Note that the MGS Web Server, included with the host software, does not support inbound SSL connections; you must use ATLAS for inbound SSL.

2. The ATLAS command in the WEBSERVICES/PARAM file has been deimplemented; the WEBSERVICES/DRIVER program will now detect being linked to by ATLAS automatically.

3. The HTTP_SERVER command in the WEBSERVICES/PARAM file has also been deimplemented; the title of the MGSWeb Web Services HTTP Server code file is now passed to the WEBSERVICES/DRIVER program via a task attribute.

4. The status of the RESPONDER program (whether running or not) has been added to the response to the DRIVER program's "Status" command.

5. WSUtility now allows you to specify the host's virtual directory name for Web Services and HTTP Services. This is used by WSUtility, along with the host name, port number and SSL option, to transfer host configuration and COBOL include files to the host. On previous releases, the virtual directory name had to be "HostWebServices".

6. The parameters to the INVOKE procedure in WEBSERVICES/LIBRARY for outbound web services calls have been changed. The first three parameters to INVOKE on previous releases (web service name, port name, and operation name) have been combined into a single "parameter array" that is generated by WSUtility and included in the application copy library. The application selects the operation it wishes to execute by specifying the corresponding operation's parameter array in the INVOKE call. The COBOL 01 name of each operation's parameter array can be set by the user via the new "Parameter Fields" tab on the "Operation Properties" page (the default name of each parameter array is "<operation name>-PARAM").

7. The field MCP-NETWORK-TIME, that was being added to the response record for all outbound web services (both Web Service and HTTP Service), has been moved to the operation parameter array as NETWORK-TIME. The old field will automatically be removed from all response records when you load the project into WSUtility.

8. A "per call" timeout value has implemented for outbound web service calls. To specify a timeout value for a web services call, the application must set the TIMEOUT field of the operation's parameter array to the timeout value in seconds. If the web services call takes longer than the timeout value, the operation will be aborted and a "transaction timed out" error will be returned.

(To be able to abort a connection attempt that exceeds the timeout value, the WEBOUT library now uses a "worker" task for each allocated socket. The worker task will be started once and reused any time that socket is re-used, unless a transaction needs to be aborted due to a timeout. If an abort is necessary, the worker task will be terminated (via a P-DS) and then will be re-started the next time the socket is re-used. Note that the worker task will not be used if the timeout value in the parameter array is zero; instead, the default socket connection timeout of three minutes will be used.)

9. A new value, called a "signature", has been implemented to allow the INVOKE function in the WEBOUT library to detect when the host configuration file for a given operation

does not match the copy library included into the calling application. The signature value for an operation is stored in the operation's parameter array, and is based on the number of fields in the operation's request and response records, as well as each field's type, length and OCCURS value. As long as the fields in the request and response records of a given operation are not changed, the signature of that operation will stay the same.

10. The following new result values can be returned by the INVOKE function on this release:

`WEB_SERVICES_DISABLEDV` (10)

Outbound Web Services calls are not enabled in your MGSWEB access key.

`HTTP_SERVICES_DISABLEDV` (11)

Outbound HTTP services calls are not enabled in your MGSWEB access key.

`REQUEST_BUFFER_ERRORV` (12)

The request buffer is too short; for COBOL applications, this error is the result of passing in the wrong request record to the `INVOKE` function.

`RESPONSE_BUFFER_ERRORV` (13)

The response buffer is too short; for COBOL applications, this error is the result of passing in the wrong response record to the `INVOKE` function.

`SIGNATURE_ERRORV` (14)

The signature value in the operation parameter array does not match the signature value from the host configuration file for the operation. This error is caused by a mismatch between the copy library included into the calling application and the current host configuration file. The most common reason for this is that the request or response records have been changed and a new host configuration file re-generated, but either the copy library was not also re-generated or the application was not re-compiled using the new copy library.

`PARAM_ARRAY_ERRORV` (15)

The operation parameter array was too short; it must be 600 bytes long.

`WORKER_ABORT_ERRORV` (16)

The WEBSERVICES/WEBOUT worker task aborted while executing the application's transaction. This error can only be returned if the application sets the `TIMEOUT` field in the parameter record to a non-zero value.

`TIMEOUT_ERRORV` (17)

The outbound web services transaction timed out and was aborted. This error can only be returned if the application sets the `TIMEOUT` field in the parameter record to a non-zero value.

11. **(The changes in this note have been backed out by version 2.01)** ~~A new "Port Properties" page has been implemented in WSUtility for Web Service and HTTP Service projects. This page allows you to set the destination "Server URL" for HTTP Service ports, and the "keep alive" option and outbound IP address the WEBOUT sockets should use for both Web Service and HTTP Service ports.~~

~~Also, HTTP Service projects can now have multiple ports declared, and those ports are now visible in WSUtility's Navigation Tree. This is to allow a single HTTP Service project to declare the same operations on different ports with different server URLs. For example, you can now declare the same set of operations on a "Production" port and a "Development" port, with the Production port's server URL pointing to the production system and the Development port's server URL pointing to a development system.~~

~~To aid in the management of multiple ports with the same list of operations, WSUtility now supports a "Clone port" function for HTTP Service projects. If you select a port in the Navigation Tree and click the *Edit | Clone port* menu item, or right-click on the port in the Navigation Tree and click the *Clone port* item on the popup menu, a new port will be created that is a clone of the selected port, including all of its settings and operations.~~

12. The MGSWeb host software will be compiled with the 51.1 compiler as of July 6, 2007.

## I.3 Release 2.01

1. The INTERFACE library has been enhanced to prevent multiple copies of the DRIVER program from running at the same time (when started from the same set of code files).

2. The SERVER program has been patched to prevent it from terminating when any AX command is entered – it now requires the AX STOP command before it will terminate.

3. WS-Out and HTTP-Out now allow the calling application to specify a server URL when calling the `INVOKE` procedure; see the `SERVER-LOCATION` field in the *Parameter Record Fields* section of this document for details.

4. WS-Out and HTTP-Out now support "Basic" HTTP Authentication; see the `SERVER-USERNAME` and `SERVER-PASSWORD` fields in the *Parameter Record Fields* section of this document for details.

## I.4 Release 2.02

1. WS-Out now supports SOAP "Header" fields on this release. Also, WS-Out support for element attributes has also been implemented (HTTP-Out already supported element attributes).

2. On this release, WSUtility now uses the "maxLength" and "totalDigits" XML schema restrictions to set the length of string and numeric fields (rather than using the default lengths of 50 characters for string fields and eleven digits for numeric fields).

3. If a web service was using the "document/literal" message style, and the request message contained elements from multiple namespaces, the WEBOUT library was not building the request XML correctly (only the outer-most namespace was included in the message). This has been corrected; the WEBOUT library will now include all required namespaces in the request XML. You must re-generate the host configuration file for all affected web services using WSUtility v2.02 to correct this problem.

4. The maximum field length for numeric fields has been increased from eleven digits to 23 digits on this release. The default length for numeric fields when importing a WSDL into WSUtility is still eleven digits, however.

## I.5 Release 3.00

1. SOAP Header field support has been implemented for WS-In Program transactions on this release. WSUtility has been enhanced to allow you to add fields to the SOAP Header or to the SOAP Body of the input and output messages. (SOAP Headers are not supported, nor required, for inbound Station or Database transactions.)

2. WS-In Program transactions have been enhanced to allow field groups and field group arrays. On previous releases, the Program transaction input and output records had to be flat structures. On this release, you can add group fields to the input and output records. Group fields can contain one or more child fields (which also can be group fields), and can occur more than once (they can be arrays).

3. The Program transaction header fields have been enhanced on this release. The IP address field in the transaction input header has been increased from 18 to 42 characters to allow for IPv6 addresses, optional SSL client certificate Subject and Issuer fields have been implemented, and the transaction output header has been reduced to six bytes (containing only the transaction token). A Program project can still be configured to use the old style 60 character headers, however (old style headers will be deimplemented in a future release).

4. WSUtility can now optionally generate a COBOL copy library for a Program project. You can use the COBOL copy library when writing the COMS application program to handle the inbound Program transactions.

5. WSUtility now displays Program transactions input and output records in a COBOL 01 record format, and lets you specify both XML and COBOL names for each field. The COBOL names are used when generating the COBOL copy library for the Program project. Also, WSUtility now also displays the total number of bytes in each input and output record.

6. WSUtility now allows you to specify an SSL client certificate it can use to communicate with the MCP system (to upload host configuration and COBOL include files). A valid client certificate is required if ATLAS is configured to require a client certificate for the port being used for the MGSWeb application. (The specified client certificate will only be used by WSUtility.)

7. WS-Out and HTTP-Out now allow you to specify a client certificate for MGSWeb to use when connecting outbound to a web services server that uses two-way SSL. You specify the name of the client certificate key in a new field called KEY-NAME in the operation parameter record. (The key must have first been stored in the Socket Keys container in Unisys Security Center.)

8. A new "Force MCP application to provide a server URL" option has been implemented for WS-Out and HTTP-Out web services. If this option is set, applications must provide a server URL in the SERVER-LOCATION field of the operation's parameter array when calling the INVOKE procedure (the default server URL stored in the host configuration file is ignored).

## I.6 Release 3.01

1. WS-In Program transactions can now contain fields with the following new data types (in addition to group, alpha and numeric fields):

- **base64**: The field will contain data encoded with the base64 encoding algorithm. The DRIVER program will automatically convert the data from base64 to binary, and the converted binary data will be inserted in the record passed to the COMS application. The field length is still limited to less than 64K bytes.

- **stream**: When the input data is received, all data from each field with this type will be converted from ASCII to EBCDIC and written directly to a stream file on the MCP system. In the request record passed to the COMS application, the field will contain the stream file title (instead of the data itself). It is up to the application to process the data in the stream file and remove the file if necessary. “stream” fields are 100 characters long in the request record (long enough to contain the file title), but the inbound data can be of any length.

- **base64Stream**: This type is a combination of the **base64** and **stream** types; the input data for a field of this type will be converted from base64 to binary and written directly to a stream file. The field will contain the title of the stream file (instead of the data itself) when the record is passed to the COMS application. It is

up to the application to process the data in the stream file and remove the file if necessary. "base64Stream" fields are 100 characters long in the request record (long enough to contain the file title), but the inbound data can be of any length.

These new field data types can only be used with fields in the Request record; they are not allowed in the Response record.

The stream files will be created under the same usercode as the MGSWeb code files, in the `WEBSERVICES/TEMPFILES/=` directory. By default, they will be stored on the same family as the MGSWeb code files. They are created EXTMODE set to OCTETSTRING.

2. A new WEBSERVICES/PARAM command, **STREAM_FAMILY**, has been implemented to allow you to specify the family on which "stream" and "base64stream" stream files will be created. By default, inbound stream files will be created on the same family the MGSWeb code files are on.

3. Because of possible naming conflicts, COBOL include files created by WSUtility for WS-Out and HTTP-Out projects will now be named:

```
WEBSERVICES/USERFILES/WEBSERVICE/<name>
```

And COBOL include files created by WSUtility for WS-In Program projects are named:

```
WEBSERVICES/USERFILES/PROGRAM/<name>
```

## I.7 Release 3.02

1. WSUtility now properly imports WSDL files that use either schema or WSDL "import" statements. Previous releases of WSUtility either ignored the "import" statements, or possibly faulted.

2. When importing a WSDL, WSUtility now correctly handles non-group schema elements that have one or more attributes associated with them. WSUtility now generates the correct COBOL source code for this type of element, and the MCP software generates correctly formed XML for them.

3. The host software now resets the transaction timeout timer for inbound Program transactions each time additional data is received from the web server. This allows inbound Program transactions to contain very large amounts of "base64Stream" data without timing the transaction out.

4. On previous releases, it was possible for the timing out of a transaction to cause a fault in the MGSWeb host software. This has been corrected.

5. WSUtility now correctly handles the "parts" attribute in the "soap:body" element in the "binding" section of an imported WSDL.

6. WSUtility will now ignore any “documentation” nodes within the “message” section of an imported WSDL.

7. The MGSWeb software can only handle importing WSDL operations that use the “request/response” transmission primitive. This restriction will be documented in the MGSWeb Reference Manual.

8. When importing a WSDL containing an operation that declares more than one “soap:header” element for an operation in the “binding” section, WSUtility will now generate a warning message that only the first “soap:header” element for the operation will be processed. This is not a change of behavior; only the warning message is new.

9. A “include field length information” option has been implemented for Window, Program and Database projects. If this option is checked, field length information will be included in the generated WSDL file. Note that this option can greatly increase the size and complexity of the generated WSDL.

10. WSUtility and the host software can now handle much larger imported WSDLs. Previous releases were limited to 256 elements per message; this limit has been increased to 2,048 elements per message. This fix also allows WSUtility to generate correct COBOL include files for heavily nested elements.

11. WSUtility now automatically saves the modified COBOL names, field lengths and OCCURS values for all elements before reloading a WSDL. This allows you to reload a modified WSDL without losing all of your changes. Modified COBOL names (that are different from the default COBOL name WSUtility generated from the XML name) are also now displayed in blue in the field list.

12. WSUtility can now handle empty complex types (a complexType with no elements) in the XML Schema when importing a WSDL. These types are converted to single-character strings in the COBOL record (PIC X). Also, other enhancements have been made to handle larger, and more complex, WSDLs.

13. The MCP Header “transaction name” field in the Program transaction request record has been increased from 18 to 36 characters, and a new numeric “transaction ID” field has been added. (The transaction ID value can be used by Program transaction applications to detect which transaction is being submitted by the client.) Also, WSUtility now allows you to set the transaction ID value for all transactions. **Note that this change requires you to regenerate all Program project host configuration and include files, and recompile your Program transaction applications.**

14. On previous releases, if the SOAP Body of the response message in an imported WSDL contained a single element that had data associated with it, the WebOut library would generate an erroneous “Unexpected data” error when the XML response was received. This has been corrected.

15. WebOut Web Service projects now support elements of type “anyType” in both the request and response records.

An “anyType” element is defined to be: an element declared in the WSDL with no defined type or a type of “anyType”; an element whose type declared in the WSDL’s schema has no type or a type of “anyType”; or, an element whose type declared in the WSDL’s schema has a single child “any” element.

For “anyType” fields in the request (outbound) record, the WebOut library will move the data from the client record to the request XML without translation – this allows the application to provide raw XML that will get copied to the request XML. (It is up to the application to ensure that the XML is valid – invalid XML in an “anyType” field can cause the server to reject the request.)

For “anyType” fields in the response (inbound) record, the WebOut library will transfer the element’s child XML (or the element’s data if the element has no child elements) from the response XML to the field in the response record. It will be up to the application to process the XML in the response record field. (Note that the copy of the XML is not a “raw” copy; whitespace, namespaces and namespace prefixes are eliminated to make it easier for the receiving application to scan the XML.)

**The WSDL must be re-imported into WSUtility, the host configuration and include files must be re-generated, and you must re-compile your applications for this feature to work.**

16. WSUtility can now handle the “attributeGroup” element in an imported WSDL.

17. WebIn now supports Program project transaction names up to 150 characters long.

18. WSUtility now uses a 90 second timeout (up from 30 seconds) when uploading host configuration and include files to the MCP host.

19. The WebOut library now converts elements of type "decimal" to a REAL in the COBOL record (previous release incorrectly converted “decimal” elements to an integer field in the COBOL record).

## I.8 Release 3.03

1. The WebOut library now supports records up to one megabyte (up from 64KB on previous releases).

2. WebIn now supports sending stream data in the response record (the **base64**, **stream** and **base64Stream** field types are now supported in the response record). For the **stream** and **base64Stream** field types, the application must insert the title of a stream file into the field; WebIn will then insert the contents of the file into the response XML data.

3. Finer trace granularity has been implemented - you can now turn tracing on for a specific WebIn or WebOut project. This is in addition to the global trace toggle. Also, when you turn tracing on using v3.03 WSUtility, only transactions against the currently loaded project will be traced. In addition, the meaning of the TRANSACTION-TRACE

item in the WebOut library's parameter record has changed; if this field is set to a non-zero value, the transaction will be traced to the WEBSERVICES/LOG file, not to the RESULT-STRING parameter.

4. The "anyAttribute" element is now supported when importing a WSDL. This allows applications to supply raw XML attributes to an element if the element is defined in the WSDL with an "anyAttribute" declaration.

5. On previous release of the WebOut library, if an integer or REAL field in the request record contained either all spaces or a zero value, the corresponding element in the generated XML did not include the value (for example, the XML for a field named "fieldOne" that contained a zero value would be "*<fieldOne />*"). On this release, zero values in numeric fields are included in the generated XML (that is, "*<fieldOne>0</fieldOne>*"). (The behavior for integer or REAL fields that contain all spaces has not changed – no value will be included in the XML.)

6. Both WebIn and WebOut now generate signed integer fields – previous releases generated "PIC 9" COBOL fields for all integer types, but this release generates "PIC S9" fields in both the request and response records.

7. The processing of the WEBSERVICES/PARAM file has been moved from the WEBSERVICES/DRIVER program to the WEBSERVICES/INTERFACE library. This allows the WebOut library to access values set in the parameter file. This fixes the problem from previous releases where, if the trace option was enabled in the parameter file but WEBSERVICES/DRIVER was not running, the WebOut library would not generate tracing information.

8. Stream files received by WebIn into the WEBSERVICES/TEMPFILES/= directory (generated by fields with the **stream** or **base64Stream** field types) will now be automatically removed by WEBSERVICES/DRIVER 15 minutes after they are last accessed by the receiving application. A new parameter file command, "*TEMPFILES_LIFE = <minutes>*" allows you to control how long the stream files remain on disk (setting this command to zero disables the automatic removal of the files).

9. WSUtility has been enhanced to prevent items from being "hidden" in the generated COBOL records. An item would be hidden (that is, not accessible) if its name matches the name of a child of one of the group items that contain it. If an item would be hidden, WSUtility now adds a number to the end of the COBOL name to make it unique.

10. WebOut projects support for the "minOccurs" WSDL option has been implemented. On previous releases, WebOut projects always included all XML elements, even if they were empty (all spaces). On this release, WebOut projects now only includes empty elements if their "minOccurs" value is greater than zero. For arrays of elements, WebOut projects will now include at least the "minOccurs" number of elements; it will include more, up the OCCURS value set in WSUtility, if they are non-blank.

11. A new WEBSERVICES/PARAM file command, ALLOW_UPLOADS, allows you to prevent WSUtility from uploading new host configuration and include files to a host.

## I.9 Release 3.04

1. MGSWeb now supports different character sets; previous releases only supported the native MCP EBCDIC character set. (Only 8-bit MCP character sets are supported; multi-byte character sets, like ASTUTL, are not supported by MGSWeb.) A new parameter file command, CCSNUMBER, has been implemented to specify an alternate character set number. The character set numbers are listed under the EXTMODE attribute in the *File Attributes Programming Reference Manual* (not all character sets listed are supported).

2. The DRIVER program now validates that WSUtility's version matches its own when uploading host configuration and include files.

3. A new parameter file command, ENCODING, supports setting the XML "encoding" value; the default encoding is "utf-8". The ENCODING value also used in the WebOut library's "Content-Type" HTTP header (for example, "Content-Type: text/xml; charset=<encoding>").

4. DMSII field names are now converted to more friendly XML names on this release (rather than all uppercase COBOL-like names). For example, the DMSII field name "FIELD-ONE" would be converted to the XML field name "fieldOne".

5. The "mcpElapsed" field value for WebIn transactions has been enhanced to include the time it takes to build the response XML.

6. For rpc/encoded operations, it was possible for a WebOut project to include multiple instances of the XSI and SOAP Encoding (soapEnc) namespaces in the <Envelope> element of the generated SOAP envelope. This has been corrected.

## I.10 Release 3.05

1. MGSWeb has been enhanced to support IPv6 networks. IPv4 and IPv6 are supported for both inbound and outbound transactions, as well as by WSUtility for uploading host configuration files and include files to the MCP system. (MCP systems support IPv6 starting with the 53.1 system release.)

2. If an inbound Program transaction timed out, it was possible for WEBSERVICES/DRIVER to get an INVALID OPERAND fault @ 38314200 or an INVALID INDEX fault @ 00033700 or 00142200. This has been corrected.

3. WSUtility has been enhanced to handle namespaces declared in the same node as they are used when reading a WSDL file (for WebOut projects).

4. WSUtility and the host software have been enhanced to properly handle empty <message> elements (that is, messages that have no <part> child elements) when importing a WSDL for WebOut projects. Previous version of WSUtility would treat operations with empty input or output messages as invalid.

5. WSUtility now displays WebOut project operations that have invalid COBOL names in red in the Navigation tree. This makes it easier to find the COBOL names you need to fix before you can generate a host configuration file.

6. WebOut projects now set the length of SOAP arrays in rcp/encoded requests to the actual number of non-blank array elements in the COBOL record, rather than setting the length to the array length set via WSUtility and including null array elements to fill out the array. This change is because some web servers can not handle null SOAP array elements with the "xsi:nil" attribute set to true.

7. Because some web servers use an HTTP result code of 500 to return SOAP faults, the WebOut library has been enhanced to process the response XML data when the HTP result code is either 200 (OK) or 500 (Internal Server Error).

## I.11 Release 3.06

1. The WebOut library has been enhanced to automatically handle "base64Binary" fields. Each element in an imported WSDL that is defined as base64Binary can have its field type set to either "base64" or "base64Stream" (the default type is "base64").

The data in a request COBOL record for "base64" fields will be automatically translated by the WebOut library to base64 before inserting it in the request XML, and "base64" fields in a response record will be automatically translated from bas64 before inserting it in the COBOL response record.

If a request record field's type is set to "base64Stream", the application should store the name of a stream file in the field; the WebOut library will read the data from the file, translate it to base64, and insert the base64 data into the request XML. For response fields that are "base64Stream", the WebOut library will translate the data from the response XML from base64, write it to a stream file, and store the title of the stream file in the field in the response record.

2. WebIn no longer requires the "mcpType" attribute for "base64," "base64Stream" and "stream" fields. This attribute will not be included in generated WSDLs, and it will be ignored if included in the request XML. WebIn now uses another mechanism to detect "base64," "stream" and "base64Stream" fields in the request XML.

3. WSUtility now sorts the list of operations for each port when importing a WSDL, to make it easier to find a given operation when the WSDL contains a lot of them.

4. A new WebOut project parameter record field, HTTP-HEADERS, has been implemented. This 150-character field allows callers of the WebOut library to specify custom HTTP headers to be added to the outbound request.

5. It was possible for a WebIn WORKER stack to get an "INVALID OPERATOR" fault after one or more Program transactions timed out. This has been corrected.

6. WSUtility has been enhanced to correctly import a WSDL that contains a request record that has multiple normal child elements and an “any” child element. Previous releases would discard the “any” child element in this case; the new release adds a “REQ-VALUE” field to the end of the record so the application can specify raw XML to replace the “any” element. Note that this feature is not allowed in response records.

7. It was possible for WSUtility to get a “Range check error” exception when opening an existing Program project file. This has been corrected.

8. If a “base64,” “stream” or “base64Stream” field occurred in a transaction in a Program project, and that transaction also had multiple SOAP header elements, all attempts to call that transaction resulted in an “Unknown transaction name” SOAP fault. This has been corrected. Also, “base64,” “stream” and “base64stream” fields are not allowed in SOAP header fields; WSUtility has been updated prevent these field types in the SOAP header.

9. The maximum XML elements MGSWeb can handle in a single document has been increased to 8,192 elements.

10. WSUtility now displays the build number from the version information stored in the code file when it is compiled.

## I.12 Release 3.07

1. It was possible for the WebOut library to leave an internal lock within the socket structure block in a locked state if the call was timed out and the destination server location was a domain name (not an IP address). This would cause any subsequent use of the same instance of the socket structure block to also time out. This has been corrected.

## I.13 Release 3.08

1. **MGSWeb Authentications Extensions** has been implemented as part of the MGSWeb **WS-Out** and **HTTP-Out** capabilities. Support is now provided for both the Windows NTLM and HTTP Digest authentication schemes (this is in addition to the existing HTTP Basic authentication capability already implemented in release 2.01).

The WebOut library will now use the “WWW-Authenticate” HTTP headers returned by the web server to automatically decide which authentication scheme to use. If the server supports multiple schemes, the WebOut library will first use Basic, then NTLM, and lastly Digest authentication. See Section 10.4, Parameter Record Fields, SERVER-USERNAME and SERVER-PASSWORD fields for more information about this feature.

## I.14 Release 3.09

1. It was possible for applications to hang inside the WEBOUT library if a fault occurred while installing a new or updated web service host configuration file. This has been corrected.

2. It was possible for the DRIVER program to fault with an INTEGER OVERFLOW error @ 56000000 when using the Atlas web server. This has been corrected.

3. MGSWeb now allows uppercase “TRUE” and “FALSE” as synonyms for “true” and “false” in XML documents.

## I.15 Release 3.10

1. If the COMS send failed when RESPONDER was attempting to send the request to the WebIn application, it was possible for the response buffer that was returned to DRIVER to be formatted incorrectly. This would cause a DRIVER WORKER fault. This has been corrected.

2. When a request timed out because the WebIn application did not respond within the “request time out” value (which is 30 seconds by default), the associated INTERFACE library request buffer was not deallocated. This would cause the INTERFACE library’s in-use memory to grow over time. This has been corrected.

3. The INTERFACE library has been enhanced to better detect if a WebIn application did not copy the first six bytes of the request buffer to the response buffer before sending it back to the RESPONDER program. Also, if the global tracing option is enabled, the INTERFACE library will now trace its activity to the WEBSERVICES log file.

4. This host release works with v3.09 of WSUtility.

5. WebIn now logs all error messages, and the messages have been enhanced to include the project name and the specific operation that failed.

6. Due to a scanner bug in WEBSERVICES/DRIVER’s AX handler, you could not enable tracing for a project whose name contained a hyphen. This has been corrected.

7. Use the patched (Unisys expression handling bug) MCP 14 compiler for software generation.

## I.16 Release 3.11

1. The Weather sample web service had moved so the URL has been updated in the project and the host file reloaded.

2. When using a client-side SSL key and a non-zero TRANSACTION-TIMEOUT value, it was possible to get a SSL error when calling the WebOut INVOKE procedure. This was caused by the worker stack that implements the timeout logic running under the usercode of a previous caller when attempting to use the supplied SSL key. This has been corrected; the worker will now change its usercode to the caller's usercode before using the SSL key in this situation.

3. A number of "library attribute error" messages, generated on each WebOut INVOKE call, were caused by WEBSERVICES/LIBRARY improperly handling LINKLIBRARY results other than 1. This has been fixed.

## I.17 Release 3.12

1. A new feature has been added to MGSWeb to allow sites to control which users can run WSUtility. Using the new USER_LIST parameter file command, a site can list the Windows user accounts that are allowed to run WSUtility. If the USER_LIST command is not in the parameter file, any user can run WSUtility; if a USER_LIST command is added to the parameter file, only the listed users can run WSUtility.

The syntax of the new command is: "`USER_LIST = <Windows user account list>;`", where `<Windows user account list>` is one or more Windows user accounts, separated by commas. Each Window user account name is limited to 24 characters, and the list is limited to 20 user accounts.

2. Additional detailed diagnostics have been added to the NTLMv1 processing routines. Debug code was added to the HTTP NTLM procedures. If one puts a value greater than zero in the TRANSACTION-TRACE field of the MGSWeb web services call parameter record, the NTLM data will be written to a separate WSBD print file.

3. A problem exists that when the COMS program, that is handling the inbound program transactions for an MGSWeb web service, does not send a response back to the RESPONDER then the program DRIVER can hang waiting for the transactions to complete while doing a host configuration file reload. This has been corrected.

4. The WEBOut facility has been enhanced to optionally support Self-Signed certificates over an SSL connection. For security reasons, when connecting to an SSL server, whose certificate is signed by a Certificate Authority running on that exact same server, the default behavior is to reject that connection. Unisys, and now MGSWeb, support an option that will ignore the fact that the received SSL certificate is self-signed and, if all other conditions are valid, the connection will be allowed.

The syntax **ALLOW_SELF_SIGNED = { TRUE | FALSE }** has been added to the WEBSERVICES/PARAM file as a systemwide control of this feature for all outbound SSL connections.

5. Web-Out now has support for accepting session cookies for a persistent HTTP connection. Via the HTTP protocol, a server can send Web-Out cookie information and Web-Out will now present the cookie information on all subsequent web services calls over that persistent connection. This facilitates servers that use a security token cookie, on persistent HTTP connections, to avoid security re-validation for every subsequent web service call over that connection. Previously, servers would have to have an HTTP re-logon before every web services call was done. This feature does not affect non-persistent (single web services call) HTTP connections.

6. When an application uses one out-bound project (Web-Out or HTTP-Out) and then calls a second out-bound project with a shorter URL specification, the URL field is not cleared before the new URL is loaded in from the WSUtility project, which results in an invalid URL specification being used for the TCP/IP connect. This problem has been corrected.