MICRO FOCUS

Xcentrisity® BIS AddPack for Visual COBOL

User's Guide

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK
http://www.microfocus.com

2020-06-17

Contents

Xcentrisity Business Information Server for Visual COBOL User's Guide 5

Copyright and Trademarks 5
Introducing the Business Information Server 5
Overview 6
Installation on Windows 7
Installation on UNIX 9
Testing the Installation 11
Uninstalling BIS for IIS 11
Uninstalling BIS for Apache 12
Using BIS 12
Web Protocols: Requests/Responses 13
Sessions 14
Tracking Sessions 14
Cookies 15
The Session Root Path and Session Scope 15
Timeouts 16
Server Response Files 17
Overview 17
Rendering Tags 18
The Rendering Process 18
Tag Options and Parameters 19
Replacement Tag Reference 22
The {{Handler}} Tag 22
The {{ContentType}} Tag 23
The {{SessionParms}} Tag 23
The {{ServiceOpts}} Tag 25
The {{ServiceArgs}} Tag 26
The {{ServiceLibs}} Tag 26
The {{StartService}} Tag 27
The {{RunPath}} Tag 28
The {{SetEnv}} Tag 28
The {{XMLExchange}} Tag 29
The {{StopService}} Tag 30
The {{SessionComplete}} Tag 30
The {{Value}} Tag 30
The {{Trace}} Tag 34
The {{TraceDump}} Tag 37
The {{Debug}} Tag 37
Control Flow Tags 40
The {{If}} / {{Else}} / {{EndIf}} Tags 40
The {{While}} / {{EndWhile}} Tags 40
The {{ Include }} Tag 41
{{//}} Comment Tags 41
Service Programs 42
Introduction 42
Service Program Lifetime 44
The XML Exchange File 44
BIS Return Codes 45
Service Program Functions 47
Server Variables Reference 66

XML Exchange Request File Format 75
Windows/UNIX Portability Considerations 78
Regular Expression Syntax 78
Metacharacters 78
Abbreviations 80
BIS Troubleshooting Tips 80
Configuring BIS/IIS after Installation 81
Command Line Configuration 81
Configuring the Run As Logon ID 82
Retrieving or Changing the Configured Identity 84
Manual Configuration 85
Setting Environment Variables 87
Setting the Maximum Thread Count 87
Notes 88
Configuration after Installation (UNIX/Apache) 88
Configuring Apache 88
Service Engine Configuration 90
xbisctl Utility 93
Creating a BIS/IIS Virtual Directory 94
Running the BISMkDir Program 94
Creating the Directory 97
Testing the New Directory 97
64-Bit Windows Considerations 97
Building and Running BIS Samples 98
Migrating from RM/COBOL to Visual COBOL 98
Using GOBACK 98
Compiling and Starting Service Programs 99
Passing Command Line Arguments 99
Retrieving the BIS_FILENAME 100
Using In-Memory Messages 100
Replacing B_Exchange 101
Using B_Trace to Write to the BIS Trace 102
Glossary 102

Xcentrisity Business Information Server for Visual COBOL User's Guide

Copyright and Trademarks

Introducing the Business Information Server

Overview

The Xcentrisity Business Information Server (BIS) is a web server environment that manages COBOL application sessions and makes them available via any web browser or other web user agent that is granted access to the BIS server. BIS offers application developers a real opportunity to build state-of-the-art Service Oriented Architecture (SOA) applications incorporating legacy business data and logic freely mixed with the latest web languages and tools.

With BIS, remote users can access data, perform application functions and execute service programs on one or multiple servers located anywhere in the world. For example, a sales force can check order status for customers during the day and enter new orders in the evening as they travel. Emergency room doctors can read patient histories on primary care physician files in another state and primary care physicians can see insurance claim's status. Bank customers can see account status, pay bills, transfer funds, and make investments, all from the comfort of their own homes. Taxpayers can have access to public records from anywhere. With BIS, any modern application architecture, function, or appearance is possible.

Xcentrisity BIS has two major components:

- A *Request Handler*, a web server extension that integrates either with Microsoft Internet Information Server (IIS) or the Apache web server.
- The *Service Engine*, which executes COBOL code under the control of the Request Handler.

A *service program* is the application COBOL code that is executed by the Service Engine. This is application dependent, and provided by the application developer.

In the simplest case, an end user enters a URL into a web browser that specifies a specific web page on a server. The web browser then formats the request using HTTP or HTTPS and sends the request to the server specified in the URL. If the requested page is a reference to a simple HTML file (usually denoted by a file extension of `.htm` or `.html`), the contents of the HTML file are sent to the browser without any further processing.

However, if the reference is to a BIS *stencil* file (usually denoted by a file extension of `.srf`), the file is read and processed by the server before it is sent to the browser. Specifically, BIS interprets the file, processing any *tags* embedded in the file's HTML or XML content. A tag is composed of text surrounded by `{{` and `}}` sequences, and tags may be interpreted as processing instructions or placeholders that are replaced by plain text, HTML or XML that is generated by the BIS service engine or by the BIS request handler.

Some useful definitions:

User Agent / Client	The program that is used to request information from a server. This program is frequently a web browser, but it could be any program on the user's machine.
HTTP	Hypertext Transport Protocol, a standard encoding scheme used to transmit requests to web servers and receive responses from web servers. HTTPS is a secure version of HTTP.
URL	Uniform Resource Locator, the location of a resource on the internet. A URL consists of a scheme (in this context, HTTP or HTTPS), the name of a machine, and a path to a file. For example, `http://microfocus.com/bis/index.html` specifies the file named `index.html` from directory `bis` on server machine `microfocus.com` using the HTTP scheme. When this is typed into a web browser, the browser issues a `HTTP GET` request on this file.
Request	An HTTP packet that contains a command issued by the user agent. A request may simply `GET` a file from a web server, `PUT` a file to the web server, `DELETE` a file from

	the web server, or may POST data (such as a form) to the server, or it may cause a program to be run on the server. GET and POST are by far the most frequently used commands.
SOAP	SOAP (Simple Object Access Protocol) is an XML-based web protocol designed to operate on HTTP to facilitate web services. It is particularly well suited to Remote Procedure Call (RPC)-style services.
REST	REST (Representational State Transfer) is an architectural style for distributed hypermedia systems and can be used to implement web services. While there is not a formal standard like SOAP, it is based on the four principle HTTP request types (GET, PUT, POST and DELETE), and URLs. In a REST architecture, a request payload may be in any format desired, including XML.
Web Server	A program that runs on a server and listens for HTTP requests. When a request is received, the web server processes the request or sends it on to another program (such as BIS) for processing. The two most common web servers are Microsoft's Internet Information Server (IIS), which BIS supports on Windows, and Apache, which BIS supports on UNIX.
Response	A HTTP packet that contains the response to the request. The response may be text, to be displayed in a web browser, or data encapsulated by SOAP for consumption by the requesting program.
Session	Requests are *stateless*, that is, the web server processes each request as if it had never received a previous request from the same user agent. A session is a BIS concept that allows sequential requests from the same user agent to be grouped together and preserves state information across requests on the server.

For more definitions, see the *Glossary*.

Installation on Windows

This covers installation of Business Information Server on Windows. Installation on UNIX is described in *Installation on UNIX*.

Prerequisites

These are the prerequisites for BIS for Microsoft Internet Information Server (IIS) running on Microsoft Windows. A host machine running one of the following operating systems is required:

- Windows Server 2008 (32-bit and 64-bit)
- Windows Server 2008 R2 (64-bit)
- Windows 7 (32-bit and 64-bit)
- Windows 8 (32-bit and 64-bit)
- Windows Server 2012 (64-bit)
- Windows 8.1 (32-bit and 64-bit)
- Windows Server 2012 R2 (64-bit)
- Windows 10 (32-bit and 64-bit)

In addition, Internet Information Server (IIS) must be installed before BIS can be installed. The procedure for installing IIS is dependent on the version of Windows.

When BIS is installed on certain non-server operating systems, such as Windows 7, there are connection limit restrictions that prevent use as a real-world web server. These systems, however, do work well for BIS/IIS application development and testing.

1. Go to **Start** > **Control Panel** > **Programs and Features**.

1. Click **Turn Windows Features On and Off**.
2. From the **Windows Features** dialog box, as a minimum, make the following selections:

- Internet Information Services
  - Web Management Tools
    - IIS Management Console [1]
  - World Wide Web Services
    - Application Development Features
      - ASP.NET [2]
      - ISAPI Extensions
      - ISAPI Filters
    - Common HTTP Features
      - Default Document
      - HTTP Errors
      - HTTP Redirection
      - Static Content
    - Health and Diagnostics
      - HTTP Logging
      - Logging Tools
      - Request Monitor
    - Performance Features
      - Static Content Compression
    - Security
      - Basic Authentication
      - Request Filtering
      - Windows Authentication

Note that other features may also be required, but are selected by default.

Once configuration is complete, close the Internet Information Server (IIS) Manager window, and reboot if required.

Installation

Business Information Server is installed separately from Visual COBOL. Install the version of Visual COBOL to be used with Business Information Server before installing Business Information Server.

---

[1] Previous versions of BIS required that **IIS Metabase and IIS 6 configuration compatibility** also be selected. This was required by the BISMKDIR configuration utility. This version of BIS includes a new configuration utility, BISMKAPP, that only works on IIS 7 and later. It has all the capabilities of BISMKDIR, but if BISMKDIR (which is still included with the installation) will be used, be sure to also select IIS Metabase and IIS 6 configuration compatibility.

[2] Selecting ASP.NET is a fast-track way of selecting most of the other prerequisites. However, it is not part of the minimal set required to run BIS.

Installation on UNIX

Prerequisites

BIS on UNIX supports the following machine environment:

- x86 running Linux (glibc 2.x) 32-bit and 64-bit
- PowerPC running AIX (6.x or 7.x) 32-bit

The Apache 2.4 web server must be installed. In order for Apache to use the BIS Request Handler, it must have shared object support. If downloading from a binary installation, make sure that it is configured with shared object support (`mod_xbis_so`). After downloading the binary installation, follow the supplier's instructions for installing Apache. If your system does not have Apache installed, or you wish to download and install the latest version, go to *http://httpd.apache.org* for more information.

Installation

BIS for Visual COBOL is installed on UNIX as an AddPack. It adds files to an existing installation of Visual COBOL. Perform the following steps as root.

1. Enter the following to ensure the BIS AddPack is executable:
   ```
   chmod +x setup_bis_mfcobol_2.1_redhat_x86_64
   ```
2. Execute the AddPack. Be sure to specify the location where Visual COBOL is installed, if it is not the default location. For example:
   ```
   ./setup_bis_mfcobol_2.1_redhat_x86_64 -installlocation="/opt/microfocus/
   VC21"
   ```

Following installation, there are a couple of configuration scripts that need to be run to complete the installation of BIS and cause it to execute whenever the UNIX machine boots.

Before running the configuration script, create a new UNIX user account or choose an existing UNIX user account to run the BIS service engine. This account must have the permissions necessary to access the data files for the COBOL program. After choosing this user account, login as root and change directory to the `bin` directory within the installation directory and run the `config_bis_daemons` shell script.

Once the configuration process is complete, the `config_bis_daemons` script displays the appropriate command to start the service daemons manually. This command is only necessary if you want to start the service daemons without rebooting the machine.

The second configuration script to run is `config_bis_apache`. This script creates a configuration file named `mod_xbis.conf` that can be given to Apache to load the BIS Request Handler. On some distributions of Apache, all that is necessary is to copy this file to the Apache's `conf.d` directory. On others, you must edit the Apache `httpd.conf` file and insert an include directive to the `mod_xbis.conf` file. When you run the `config_bis_apache` script, it displays further instructions on installing the configuration file. Redirect standard output to a file to refer to these instructions later, or use the copy feature of your terminal emulator, if it supports that. See *Configuring Apache* for more details on configuring Apache.

After Apache has been successfully configured, either start or restart it for the configuration to take effect.

Configuring BIS

After the AddPack is installed, the BIS Request Handler and Service Engine must be configured.

The first decision you must make is whether you want to run the 32-bit or 64-bit version. The "bitism" of the Request Handler must match the "bitism" of the Apache web server. That is, both must be 32-bit programs or both must be 64-bit programs. Similarly, either the Request Handler and the Service Engine must both be 32-bit programs or else both must 64-bit programs.

The Service Engine runs Visual COBOL programs. Although the Service Engine runs as the root user, the COBOL programs normally do not run as root. When you configure the Service Engine, you specify the user ID used to run COBOL programs. You can create a new user ID for this purpose, or you can use an existing user.

Configuring BIS Service Engine

Use the `config_bis_daemons` script (in the bin directory) to configure the Service Engine. The UNIX Service Engine is implemented as two UNIX daemons. They both are started and stopped by a single `init` script. They both read the same configuration file, normally named `/etc/xbis.conf`. The script accepts several parameters:

-b *bits*	*bits* is either 32 or 64, meaning the 32-bit or 64-bit version of BIS
-e	Edit the `xbis.conf` file by interactively asking questions
-h	Print this message
-q	Show brief messages, rather than the more verbose messages

If you are familiar with this process, edit the resulting `/etc/xbis.conf` file. Otherwise, you can specify the -e parameter and let the script help you.

Running the script creates a file named `xbis.conf` and places it in the `/etc` directory. It also places the init script which starts and stops the BIS daemons in the appropriate place in directories under `/etc`. The script will tell you the command to run to start the BIS daemons. The BIS daemons will also start automatically when UNIX restarts.

Note that the user ID used to run Visual COBOL programs under the Service Engine is specified in the `/etc/xbis.conf` file. If you specify a nonexistent user ID, `config_bis_daemons` reminds you to create the user ID before starting the BIS daemons. If you do not do this, the BIS daemons do not start. This is the most common cause of failure to start the BIS daemons.

Configuring BIS Request Handler

Use the `config_bis_apache` script (in the bin directory) to configure the Request Handler for use with the Apache web server. It accepts these parameters:

-b *bits*	*bits* is either 32 or 64, meaning the 32-bit or 64-bit version of BIS
-w	The directory where BIS creates files while running.

Running the script creates a file named `mod_xbis.conf` in the `etc` directory within the Visual COBOL installation directory. It also displays instructions for how to use this file. In most cases, you need to add an `Include` directive in the Apache configuration file that points at this file, and then restart the Apache web server.

Edit the `mod_xbis.conf` file to configure Apache for your BIS applications, and also to enable debugging during development.

Configuring Apache

Configure the Apache web server so the `mod_xbis.conf` file produced by the `config_bis_apache` script is read by Apache when it starts. If your version of the Apache installation has a `conf.d` directory, place the `mod_xbis.conf` configuration file into this directory. If your version of Apache does not use a `conf.d` directory, edit the main `httpd.conf` configuration file to include the following line:

```
Include Your-COBOL-Installation-Directory/etc/mod_xbis.conf
```

Any further changes to the configuration of the Apache portion of BIS should be made to the `mod_xbis.conf` configuration file.

See *Configuration After Installation* for more information on configuring the Apache Request Handler.

Testing the Installation

The samples are the best way to verify that BIS was successfully installed. To launch the samples on the server for BIS installed on a Windows system:

- 32-bit installations: **Start** > **All Programs** > **Micro Focus Xcentricity BIS for Visual COBOL** > **Verify XBIS (x86) Installation**.
- 64-bit installations: **Start** > **All Programs** > **Micro Focus Xcentricity BIS for Visual COBOL** > **Verify XBIS (x64) Installation**.

Or you can launch the verification program from any web browser on the local machine with one of the following:

```
http://localhost/xbisvc22/samples/default.srf?trace=page
```

```
http://localhost/xbisvc22.x64/samples/default.srf?trace=page
```

To launch the samples on either Windows or UNIX, start a web browser and enter the URL:

```
http://localhost/xbisvcob/samples/MF/default.srf
```

If you installed BIS on a different machine, replace `localhost` with the name of the Windows or UNIX machine running IIS or Apache. If the web browser is running on the same machine as IIS or Apache, then localhost refers to the current machine and may be used as the host name.

You should see the **Welcome to the BIS Samples** page:

As an additional test, click on the link to the first sample, **verify**. The **BIS Verify** sample page will be displayed, which is running the VERIFYBIS service program. Follow the instructions on this page to complete the verification.

Uninstalling BIS for IIS

To uninstall BIS/IIS, use the **Programs and Features** control panel applet:

1. Click **Start > Control Panel**, and select **Programs and Features**.
2. Click the version of BIS that you wish to remove.
3. Click the **Uninstall** button.

4. When the Program and Features message box appears requesting "Are you sure you want to uninstall Xcentrisity BIS for Visual COBOL 2010?", press the **Yes** button.

Removing Only the Web Application Samples on IIS

To remove the samples from a Windows IIS web site after installation, log onto the server and then:

1. Click **Start > Control Panel > Administrative Tools > Internet Information Services**.
2. In the Connections pane, expand the machine's name, then expand **Sites**, then **Default Web Site** (or your web site, if renamed).
3. Right-click **xbisvcob** and select **Remove** from the popup menu.

On IIS version 6 and later (that is Windows 7 or Windows Server 2008 onwards), deleting the web virtual directory/application will not remove the physical folder. To complete the removal, delete the xbisvcob physical directory (usually found under `c:\inetpub\wwwroot`) using Windows Explorer or from the Windows command line.

Uninstalling BIS for Apache

To uninstall BIS/Apache, you must uninstall both the BIS Request Handler and the Service Engine. These are uninstalled separately, just as they were configured separately.

Uninstall the BIS Request Handler

When you configured the BIS Request Handler, you most likely edited an Apache configuration file to add an `Include` directive that points to the BIS `mod_xbis.conf` file. To uninstall, just edit the same Apache configuration file and remove the Include directive.

You will then need to restart or shut down the Apache server. This is commonly done by means of the Apache `apachectl` command.

Uninstall the BIS Service Engine

To uninstall the BIS Service Engine, you must first stop the Service Engine. The Service Engine is implemented as a pair of UNIX BIS daemons. The `config_bis_daemons` script contains name of the command used to start the daemons. Use the same command to stop the daemons. This command is a shell script named `xbisengd`. Include a single `stop` parameter with this command.

It is located in a subdirectory of `/etc`, typically `/etc/init.d` or `/etc/rc.d/init.d`, depending on the version of UNIX or Linux.

A typical command would look something like this:

```
/etc/rc.d/init.d/xbisengd stop
```

The `config_bis_daemons` script which configured the Service Engine made some changes to the UNIX system, such as installing the `xbisengd` script. It also created a script to undo these changes.

Run the `config_bis_daemons` script in the `bin` directory to uninstall the BIS Service Engine.

Using BIS

BIS functions as an extension to a web server, providing additional capabilities-namely, the ability to render and serve `.srf` stencil files, and the ability to quickly make both new COBOL programs and legacy COBOL programs available on the Web.

In order to understand how COBOL programs and the Web interoperate, some web concepts must also be understood. These are described in the next sections.

Web Protocols: Requests/Responses

Web clients and servers communicate by using a request/response protocol called HTTP, which is an acronym for Hypertext Transfer Protocol. HTTP includes two methods for retrieving and manipulating data: `GET` and `POST`.

`GET`	Retrieves data from the server. The target of the request (referred to as a *resource*) is specified as a *URI (Uniform Resource Identifier)*. This is usually (but not always) an absolute reference to a file on the server and is referred to as a *URL (Uniform Resource Locator)* when used in this context. Additional parameters, called Query Parameters, can also be specified.
`POST`	Posts data back to the server. In addition to a URL and query parameters, a POST request includes a *payload*. The payload is usually form data, the aggregated contents of the various fields (also called *controls*) that were in the response.

There are other methods (`HEAD`, `PUT`, `DELETE`), but the above two are the ones used by BIS for SOAP based web services. The other methods are available for REST-based web services.

The general form of a URL is familiar to anyone who has used a web browser:

```
http:// host [:port] / [absolute_path [ ? query_parameters ] ]
```

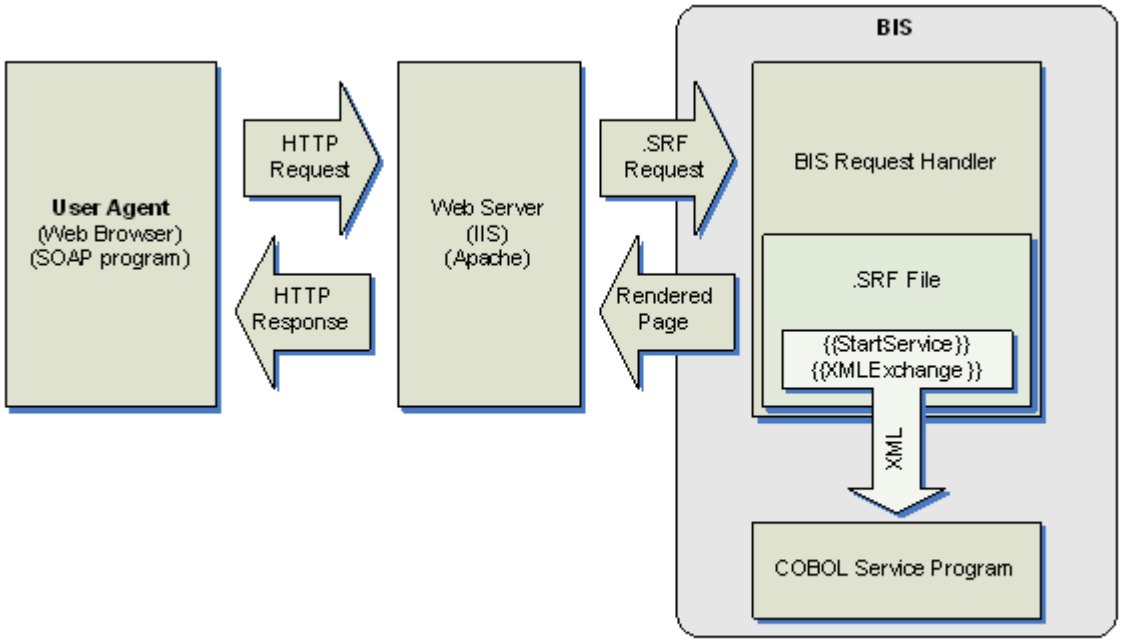
where:

`http://`	Indicates that the Hypertext Transfer Protocol is being used to make the request. In a URI, this is referred to as the scheme. BIS supports two schemes: `http` and `https` (secure http).
`host`	The name or location of the computer that will receive the request.
`port`	An optional integer that specifies the port on the server that will receive the request. If omitted, this defaults to `80` for the `http` scheme, and `443` for the `https` scheme. The combination of `host` and `port` (along with *host headers*, which is a scheme that allows a single host to serve multiple domains) specifies a unique web server.
`absolute_path`	The absolute location of the resource being requested on the host. This is frequently (but not always) the name of a file. Note that the base directory is not the root directory of the file system, but the root directory of the web tree that is being served by the host on the specified port.
`query_parameters`	Optional parameters that are made available to the web server and to the service program.

To summarize, a client (web browser or program using SOAP) sends an HTTP request to the web server. The request contains a method (GET or POST), a URI that specifies the file or resource that is being requested, optional query parameters, and optional form data (if a POST).

If the resource being requested is a resource that is associated with BIS by the web server, for example, a `.srf` file (sometimes also called a *stencil*), then all of the above information (request type, URI, query parameters, form data) is passed to the BIS Request Handler, which then renders (that is, executes) the tags in that file. If BIS renders a `StartService` tag, a COBOL service program is started. If BIS subsequently renders an `XMLExchange` tag, the request is sent to the COBOL program, and the COBOL

program's response is rendered into the HTTP response text that is returned to the user agent (browser, SOAP consumer, etc.).



Sessions

HTTP requests are innately *stateless*. The web server does not provide any built-in mechanism to group consecutive requests together. However, once a service program is started, subsequent requests from the same user agent should be routed to the same service program. To make this possible, BIS creates a Session, which is a container of information for the user agent that persists from one request to the next . A session is automatically created when a request first arrives from a particular user agent. The Session contains information that BIS uses to recognize requests as belonging to a sequence, and associates requests with persistent data and services.

A Session is automatically created when BIS receives a request that cannot be associated with an already existing session. Once a Session is created, it persists until:

1. The Session is complete: this can be requested by either the service program or by a special handler tag-the `SessionComplete` tag.
2. A predetermined but adjustable amount of time passes without an additional request from the user agent, referred to as the *Inactivity Timeout* period.

Active Sessions use server resources, and if a service program is waiting for a request, this can be significant. Because site visitors may simply close the browser window without performing any action that indicates that they are finished with the application, BIS will free those sessions and resources after a predetermined period of inactivity.

Tracking Sessions

There are three common ways for servers to implement session tracking:

1. A unique ID may be placed into the URL of subsequent pages.
2. A unique ID may be placed in the query parameter of subsequent pages.

3. The server sends a *cookie* that contains a unique identifier with the response. The user agent saves the cookie, and then includes the cookie with the next request.

BIS uses the third method, cookies, to identify sessions.

Cookies

In order to track user agent sessions, the BIS Request Handler places a Cookie in the responses that is sends to the user agent. The Cookie has a specific name, is associated with a specific URL, and contains the information that the BIS Request Handler can use to identify the session. When the client sends a request to the specific URL (or a URL containing the Cookie's URL), the client also sends the Cookie back in the request.

When the server receives this request, by default, BIS looks for a *Cookie* in the request to locate a session created by a previous request from the same user agent. When the BIS Request Handler receives a request containing the specially-named cookie, it uses the contents of the cookie to search for an existing session. If the session is located, BIS services the request using that session. If the session is not located, a new session is created for the request and the new session's cookie is included with the response.

The disadvantage of using cookies is that some user agents purposely disable cookies for privacy reasons: unscrupulous web sites can use permanent cookies to track the user agent's repeat visits over a long period of time. BIS uses only session cookies-a type of cookie that is automatically deleted when the user agent terminates-to avoid these concerns. It is also possible to configure a user agent to ignore session cookies. This will, unfortunately, prevent BIS applications from working with that user agent.

The Session Root Path and Session Scope

As stated above, when a session is created, the BIS server will include a Session Cookie that uniquely identifies the session with the response. The user agent saves the cookie, and includes the cookie with subsequent requests. The BIS server uses the cookie to associate requests with sessions.

Cookies are shared by all instances of a particular user agent. This makes it difficult for a particular user agent to gain access to more than one session on the server-if multiple browser windows on the same client machine request the same page, each window will send the same cookie, BIS will see the requests as originating in a single window, and will not create additional sessions. Multiple sessions are desirable if the end user wishes to run multiple BIS applications hosted by the same server in separate windows, or the application developer wishes to include multiple applications in a browser window by using HTML `<OBJECT>` or `<IFRAME>` tags.

Fortunately, there is a solution: the scope of a particular session cookie can be restricted to particular URL paths on the server. The user agent will only include the session cookie with a request URL that is as specific as, or more specific than the path that was specified when the cookie was stored in the user agent.

IIS and Apache derive the default application root path differently:

- *IIS* defines the virtual directory that contains the BIS application as the application root path. A virtual directory is created during the initial installation, and additional virtual directories (and hence Application Root Paths) can be created at any time, either using the IIS Administration tool or by using the `BISMkDir` utility program that is provided with BIS.
- `Apache` uses the value of the `BIS_ROOT_PATH` environment variable as the application root path. This is usually defined in the `mod_xbis.conf` configuration file and is set during installation. Multiple application root paths may be defined on a single server by defining the environment variable within the appropriate `<Directory>` section of the configuration file.

The application root path may be changed by using the `{{SessionParms}}` tag only during the rendering of the session's initial page. The session root directory may be set to:

1. `DEFAULT`: the application root path.
2. The path that directly contains the requested object.

3. Any path that contains the requested object. However, the path cannot be closer to the root directory than the application root path.

For example, if the request URL is

```
http://microfocus.com/xbis/apps/states/texas/default.srf
```

and the default application root path is

```
/xbis/apps
```

then the application root path may be changed to any one of

- /xbis/apps
- /xbis/apps/states
- /xbis/apps/states/texas

See *The {{SessionParms}} Tag* for more information.

Timeouts

BIS supports two kinds of timeouts:

- Session Inactivity Timeouts
- Service Timeouts

These timeouts are described in detail in the following sections.

Session Inactivity Timeout

Session inactivity timeouts are used to detect abandoned sessions and free server resources by deleting those sessions. For example, each active service program counts against the BIS Service Engine use count. If abandoned sessions are allowed to idle for an excessively long time, there may be a number of idle service programs consuming resources that could be recycled to handle new requests. The purpose of the session inactivity timeout is to free those resources.

To detect abandoned sessions, BIS stores the time the most recent request was received in the session. At various intervals, BIS determines if a session has been inactive longer than the timeout period set for the session. If so, the session is released.

There are two ways to indicate proactively that a session is complete and may be released:

- On the page: embed the `SessionComplete` tag.
- From a service program: call `B_WriteResponse` and specify `BIS-Response-SessionComplete` as the optional parameter.

In all cases, the session is not released until it is inactive; that is, all services within the session have ended and there are no active requests using the session.

Setting the Session Inactivity Time

The default inactivity timeout value for a BIS session is 600 seconds (10 minutes). However, the inactivity timeout value can be changed in several ways:

- The timeout value may be globally set for all BIS sessions on the server with the *BIS_SESSION_INACTIVITY_TIMEOUT* environment variable on BIS/IIS and the Service Engine InactivityTimeout option keyword on BIS/Apache. This option is set in the `/etc/xbis.conf` file. The value must be specified in seconds. For example, on Windows:

  ```
  BIS_SESSION_INACTIVITY_TIMEOUT=600
  ```

  This environment variable sets the timeout to 600 seconds (10 minutes). See *Setting Environment Variables* for information about setting and modifying environment variables on Windows, and *Service Engine Configuration* for information on configuring BIS on UNIX.

- The timeout may be set from within a `.srf` file by using the `SessionParms(InactivityTimeout=seconds)` tag (see *The {{SessionParms}} Tag*). Note that this parameter is specified in seconds and takes effect as soon as the tag is rendered.
- The service program may set the timeout with the `B_SetInactivityTimeout` call. Note that this call does not take effect until the next time the service program interacts with the BIS Request Handler; that is, the service calls `B_ReadRequest` and BIS renders an `XMLExchange` tag.

Of these, the *BIS_SESSION_INACTIVITY_TIMEOUT* variable and InactivityTimeout option keyword have the lowest priority and are overridden by either the `B_SetInactivityTimeout` call or the `SessionParms` tag.

The largest value that the session inactivity timeout interval can be set to is 1,000,000 seconds (about 11 days).

Service Timeouts

When the BIS Request Handler passes a request to a service program, page rendering is suspended while the program performs the required processing. The service timeout value sets an upper bound on the amount of time that page rendering will be suspended.

The default service timeout is 30 seconds. This value can be changed in the following ways:

- The service timeout value may be globally set for all BIS sessions on the server with the `BIS_SERVICE_TIMEOUT` environment variable on BIS/IIS and with a Service Engine `ServiceTimeout` option keyword on BIS/Apache. The value must be specified in seconds. For example, on BIS/IIS:

  ```
  BIS_SERVICE_TIMEOUT=30
  ```

  This environment variable sets the timeout to 30 seconds. See *Setting Environment Variables* for information about setting and modifying environment variables on Windows, and *Service Engine Configuration* for information on configuring BIS on UNIX.
- The timeout may be set from within a `.srf` file by using the `SessionParms(ServiceTimeout=seconds)` tag. Note that this parameter is specified in seconds and takes effect as soon as the tag is rendered.
- The service program may set the timeout with the `B_SetServiceTimeout` call. Calling this function with a parameter of `0` restarts the timer without changing the current value. This is useful as a `keep-alive` function when performing lengthy processing.

Of the above, the `BIS_SERVICE_TIMEOUT` variable and `ServiceTimeout` option keyword have the lowest priority and are overridden by either `SessionParms` tag or the `B_SetServiceTimeout` call.

Server Response Files

Overview

The Server Response File is the key control mechanism of BIS and BIS-enabled web applications and services. Each web application and service will contain at least one unique Server Response File, identified by the extension `.srf`. A Server Response File is also sometimes referred to as a *stencil*, since it acts as a stencil during the process of composing the content of an HTTP response to a request from a User Agent.

Server Response Files are can be HTML files augmented by additional information to control dynamic (program generated) content. In these cases, there are two differences between Server Response Files and regular HTML files:

- When the user agent (usually a web browser) requests a .srf file that is contained within a directory served by BIS, the web server automatically loads and activates the BIS *Request Handler* to serve the

file. A *Request Handler* is a component invoked by a web server such as Internet Information Server (IIS) or Apache to service a particular type of request; in this case, a request for a Server Response File.
- Server Response Files will normally contain additional, non-HTML *Rendering Tags* that direct BIS to perform various kinds of processing and substitution while the page is being used to render the response content. This process usually includes execution of, and interaction with, Visual COBOL-based service programs whose execution is controlled and synchronized by BIS.

If Server Response File is used with web services, the Server Response File only contains the necessary tags to allow the Request Handler to route the request to the service program implementing the web service. Care must be taken when creating Server Response Files for web services not to introduce any extra formatting into the response that will confuse the client.

Rendering Tags

Rendering tags are text strings embedded in the server response file HTML source code. A rendering tag has this general form:

```
{{ tag }}
{{ tag (parameter-list) }}
```

Rendering tags always begin with `{{` and end with `}}` sequence and the tag itself is not case-sensitive, although parameters within the tag may be case-sensitive. Spaces are used in the examples to increase readability but are not required.

The optional parameter list may be formatted in a number of ways:

- As a comma-separated list of tokens:
  ```
  {{ ServiceLibs ( xmlif, mylibrary ) }}
  ```
- As a comma-separated list of key-value pairs:
  ```
  {{ SessionParms( InactivityTimeout=600, ServiceTimeout=30 ) }}
  ```

Except where specified, tokens may be enclosed in double or single quotation marks. This is required if a token contains spaces or a comma.

Under Windows, the total length of a tag (from the opening brace to the closing brace) may not exceed 4096 characters.

Important: Important: Both the opening `{{` and the closing `}}` tag delimiters must be contained on a single line; that is, a tag may not contain embedded newline characters. Use caution when creating tags with HTML editors that reformat HTML and make sure that any reformatting does not split tags across multiple lines. Some strategies to avoid line wrapping problems:

- Turn off line and word wrapping in your HTML editor for `.srf` files. Note that Visual Studio properly handles tags within the HTML editor.
- Embed non-rendering tags (that is, tags that do not produce HTML output) in HTML comment sequences, as HTML editors will preserve formatting within comments. For example:
  ```
  {{ ServiceLibs( MyVeryLongLibraryName AnotherVeryLongLibraryName ) }}
  ```

You may still have to disable word-wrapping and reformatting for `.srf` files to prevent reformatting, or create tags that do not contain spaces.

The Rendering Process

When the user agent requests a page from the web server, and the page designates a Server Response File (that is, the file is in a directory associated with BIS and has a `.srf` suffix), the page is automatically served by the BIS Request Handler. The page is processed from top to bottom and tags are rendered as they are encountered.

There are two basic kinds of rendering tags:

- *Processing Control Tags* are tags that are completely removed from the final rendered content.
- *Substitution Tags* are completely replaced in the final content by new (possibly empty) text.

If a tag is not recognized, it is rendered literally-that is, the tag appears in the output unchanged.

Note: Tags are order-dependent. A particular tag may affect how subsequent tags are rendered; for example, the `StartService` tag specifies the service that the `XMLExchange` tag uses. In addition, the `Handler` tag must be the first non-comment tag in every file, and it must appear within the first 4096 characters of the file.

Processing Control Tags

Processing Control Tags control how the page is processed by the BIS Request Handler. There is a tag that specifies the name of the service program to run to serve the page, tags that set processing options, and tags that allow for conditional processing (for example, parts of the page may be skipped).

Processing control tags are always removed from the rendered response.

Substitution Tags

Substitution Tags are replaced with new literal text, HTML, or XML. These tags are replaced by output from the service program or by the BIS Request Handler directly without program interaction.

Tag Options and Parameters

A particular tag may have one or more options or parameters. If this is the case, the options are specified in parenthesis after the tag name, except for the Handler and Include tags.

Pathnames

There are two kinds of pathnames used within tags:

- A fully qualified pathname begins with a slash. On BIS/IIS, the slash may optionally be preceded by a drive letter specification.
- A relative pathname is any pathname that does not follow the above rules.

Relative pathnames are interpreted relative to the *current directory*. Under BIS, the current directory is the directory that contains the `.srf` file being processed.

The current directory for the BIS Service Engine is set when the `StartService` tag is executed. If a `.srf` file is subsequently served from a different directory, the current directory of the already-started Service Engine is not changed. However, any relative pathnames in the new .srf file are still interpreted relative to the directory that contains that `.srf` file.

On BIS/IIS, pathname resolution within the BIS service program is affected by the environment_mapper and program_search_order run-time tunables. These may be used to great effect in service programs in conjunction with the `SetEnv` tag.

Referencing Files in System Locations

Several techniques are provided that allow files in system locations to be referenced from within a `.srf` file.

Under BIS/IIS, the following environment variables are useful in pathnames. Note that `EXPAND_ENV_VARS` must be set in the service configuration file for these to be useful.

Variable	Description
`ProgramFiles`	The location of the Windows `Program Files` directory.

Variable	Description
SystemRoot	The drive and directory containing the Windows operating system.
TEMP TMP	The fully qualified path to the directory containing temporary files for the current process. Note that TMP and TEMP normally refer to the same directory, but this is not required.
USERPROFILE	The user's home directory.
WINDIR	Same as SystemRoot .
AllUsersProfile	The home directory for **All Users**.

On BIS/IIS, you can also define synonyms on the server using the environment_mapper run-time tunable, or directly define environment variables using the SYSTEM control panel applet:

```
Start > Control Panel > System > Advanced > Environment Variables
```

For example, if you add `MyPrograms="c:\My Programs"` to the environment, and have EXPAND_ENV_VARS in your configuration file, then you can refer to the file `abc.cob` by specifying a path of `$MyPrograms/abc.cob`. See *Setting Environment Variables* for information about setting and modifying environment variables on Windows.

On UNIX, use the `mod_xbis.conf` configuration file to define BIS environment variables. The installed `mod_xbis.conf` file contains Apache SetEnv commands to globally set variables named *COBDIR* and *SIXFOUR*. It also contains examples of environment variables in a single directory. See *Configuring Apache* for details.

Predefined BIS Environment Variables

BIS adds the following variables to the environment under both IIS and Apache. Note that these variables are dynamically set during execution and are only available in the service program. They are not visible in your shell environment or in the `.srf` file.

Variable	Description
BIS_HOME	The directory from which the BIS Service Engine is loaded. On Windows, this is typically XBIS.EXE in C:\Program Files\Micro Focus\Xcentrisity BIS for Visual COBOL .
BIS_FILENAME	The fully qualified name of the temporary file created for this session used to exchange data between the BIS Request Handler and the COBOL service program. When the COBOL service program calls B_WriteResponse, the BIS Web Server reads this file to obtain the content (XML, HTML or plain text) replacing the XMLExchange tag in the response. When the service program calls B_ReadRequest, the current web request document (XML) is written into this file. This includes any content such as the POSTed-back form variables, the request variables, and server variables, all encoded as an XML document. By default, this file is created in the Windows TEMP or, on UNIX, the BIS temp directory. This can be controlled during the UNIX installation with the "Name of BIS temp directory?" installation prompt and after the UNIX installation with the TempDir Server Engine

Variable	Description
	Configuration. Both the BIS Request Handler and the Service Engine must have permission to create, read, and write files in this directory. The BIS installation procedure adds the required permissions to this directory.

The COBDATA and COBPATH Environment Variables

If a relative filename is specified, the BIS service attempts to locate a data file by searching the directories specified by the COBDATA environment variable and a program object file by searching the COBPATH environment variable. For full details of how the BIS service program locates files, see the description of the `program_search_order` run-time tunable and "Filename Mapping" in the Visual COBOL documentation.

Troubleshooting Tags

If a tag is not performing the expected function, the tag may be malformed or may have been altered by an HTML editor. The following steps can help isolate this problem:

Is the tag itself visible in your web browser?

This indicates that BIS is not recognizing the tag. Check the spelling of the tag and be sure that the HTML editor did not split the tag across multiple lines-tags may not contain line break characters or span lines (you'll have to use the browser's **View > Source** to examine the raw HTML to be sure). On UNIX, enabling tracing (see below) and setting the `BISStencilDebug` configuration option will cause the generation of a trace message with the reason why a tag was rejected.

Did the tag fail to perform the requested function?

If a malformed tag is embedded in an HTML comment (see the example in the *Rendering Tags* section), the tag may fail to render but not be visible in the rendered output. To see such tags, use your web browser's **View > Source** command. Tags should never appear in the raw HTML that is sent to the web browser.

Does the tag appear in the trace output?

Enable tracing and examine the trace output. If you have access to the `.srf` file, to quickly enable tracing, insert this tag after the `Handler` tag:

```
{{ Trace(start,page) }}
```

Then request the page using your web browser. This will cause trace output to be appended to the end of the current page. The trace output indicates when most tags are rendered and the results of the rendering.

On BIS/IIS, to direct trace output to a file, replace `page` with `file` (or specify both using page,file). This will direct all trace output for the session into a file in the server's temporary directory (normally `C:\Windows\Temp`), or the directory specified in the trace `dir=` parameter. If you use this type of tracing, be sure to occasionally delete these files from the temporary directory.

The trace files use the following naming convention:

```
BIS-ssss-trace.txt
```

Where *ssss* are the initial characters from the session identifier. The first four non-slash characters of the session identifier are always used; if a file of that name already exists, BIS will continue to add characters from the session ID until the filename is unique.

On UNIX, trace output is directed to a file if tracing is enabled. A separate trace file is created for each session and is placed in the UNIX `/tmp` directory unless the `BISTraceDirectory` configuration option is specified or redirected with the trace `dir=` parameter. So on UNIX, Trace(start) is sufficient to create a trace file.

Note that on UNIX, the `BISMasterTrace` configuration option must be enabled before any tracing can occur. See *Configuring Apache* for details about setting or clearing this option.

Tracing is the most useful of the above techniques and should be enabled during the development process.

Replacement Tag Reference

This section presents and discusses each tag that is implemented in Server Response Files.

Here is an example of a basic `.srf` file. Tags are italicized.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/
strict.dtd">
<!-- BIS control tags (removed when page is rendered) -->
<!-- {{ handler * }} -->
<!-- {{ Trace(queryparam=trace) }} -->
<!-- {{ StartService(samp03.dll) }} -->
<html>
<head>
    <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title> COBOL Web Server Demonstration Page</title>
</head>

<body>
    <div align="center">
        <h3> COBOL Web Server Demonstration Page</h3>
    </div>
    <p>--- Begin Application-Generated XHTML ---</p>
    <div>
        {{ XMLExchange(OnExit="goodbye.srf") }}
    </div>
    <p>--- End Application-Generated XHTML ---</p>
    {{ TraceDump }}
</body>
</html>
```

Note that the first three tags in this example are embedded in HMTL comments. This is not strictly necessary from an operational standpoint (and may be undesirable because empty comments will be sent to the browser), but useful to keep an HTML editor like Microsoft FrontPage, Expression Web, Visual Studio or Adobe Dreamweaver from reformatting the text in the `handler` tag, or possibly splitting the tag across multiple lines. Some of these products support the server response file syntax directly and do not have this issue.

The {{Handler}} Tag

This tag must appear at or near the beginning of every server response file that is to be processed by BIS. It indicates that this particular `.srf` file contains Xcentrisity BIS rendering tags.

```
handler *
```

The `handler` tag's parameter indicates the name of the handler to be invoked to render the tags within the stencil, with `*` indicating the default tag handler. In this release of BIS, the only supported handler tag is the default, so `{{ Handler * }}` is the recommended format of this tag.

Future versions of BIS may support additional handlers.

Notes

- The handler tag must appear in every `.srf` file, including `.srf` files included in other `.srf` files.
- The `handler` tag must precede all other non-comment tags, and must appear within the first 4096 characters of the file. (Note that BIS/IIS allows `include` tags to precede the `handler` tag.)

- Only one `handler` tag in each `.srf` file is permitted. On BIS/Apache, multiple `handler` tags are allowed, but only the first encountered in the file is relevant.

The {{ContentType}} Tag

This tag sets the content type for the HTML response.

```
ContentType ( value )
```

BIS does not attempt to interpret the value, which encompasses the entire parameter, including commas and any quotes.

Examples

```
1. {{ ContentType(text/html; charset=utf-8) }}
2. {{ ContentType(text/xml) }}
```

Notes

- If not specified, the default content type is `text/html; charset=utf-8`. On BIS/Apache, if the content type of the request message indicates an XML message, the default content type of the response is `text/xml; charset=utf-8`.
- If `{{ContentType}}` is specified multiple times on a page, the last instance is used.

The {{SessionParms}} Tag

This tag allows various session attributes to be set:

```
SessionParms( InactivityTimeout= seconds | DEFAULT,
              ServiceTimeout= seconds | DEFAULT,
              Path = DEFAULT | path,
              Scope = ALL | ISOLATE )
```

where:

`InactivityTimeout`	Determines how long a session survives without user interaction. • `DEFAULT` - Resets the timeout to the default setting: 600 seconds or 10 minutes (on BIS/IIS) or the default inactivity timeout configured in `/etc/xbis.conf` (on BIS/Apache). • *`seconds`* - An integer that specifies the number of seconds before the session terminates. The minimum value is 10 seconds (useful for testing) and the maximum value is 1,000,000 seconds (about 11 days).
`ServiceTimeout`	Determines the maximum length of time the Service Program may run when a request is received. • `DEFAULT` - Resets the timeout to the default setting: 30 seconds (on BIS/IIS) or the default inactivity timeout configured in `/etc/xbis.conf` (on BIS/Apache). • *`seconds`* - An integer that specifies the number of seconds before Service Program termination processing begins. The minimum value is 10 seconds (useful for testing) and the maximum value is 3,600 seconds (one hour).

`Path`	Specifies the root path of the current session. This parameter is ignored unless the page being served is the initial session page.
	• `DEFAULT` - The session root path is set to the path of the current request. See *The Session Root Path and Session Scope* for more details.
	• *`path`* - Explicitly sets the session root path to path. The `path` may be specified as a relative or an absolute URL path and must specify a path segment contained in the URL path of the initial page.
	In addition, under IIS, the specified path must contain at least as many path segments as the application root path (the base directory for the BIS application)- that is, the path cannot be closer to the root of the web than the BIS virtual directory.
	For example, if the requested page is
	`http://microfocus.com/xbis/apps/states/texas/default.srf`
	the default session path is
	`/xbis/apps/states/texas`
	These paths may be specified to set the session root path to xbis/apps:
	`Path="/xbis/apps"`
	`Path=../../`
	These paths set the default session path to the directory containing the requested object:
	`Path=DEFAULT`
	`Path=.`
	`Path=”/xbis/apps/states/texas”`
	These paths are invalid and are reported as being invalid in the trace file:
	`Path=/xbis/apps/states/california`
	`Path=../florida`
	These directories are not contained in the path.
	In addition, for IIS servers, the path cannot be closer to the root than the application base path (the path that describes the virtual directory that contains the BIS server).

`Scope`	Determines the scope of the current session. This parameter may be specified at any time and is not inherited by new sessions. • `ALL` - All pages served from the session base directory and subdirectories of the session base directory are served as part of the current session. This option is the initial default for all new sessions and is appropriate for most applications. • `ISOLATE` - Only pages served from the session base directory are included in the current session. A new, non-isolated session is started when a page is requested from a subdirectory of the session base directory. The `ISOLATE` option allows a single user agent to use more than one BIS session as long as

	the sessions are based in separate directories on the server.

Notes

- All parameters are optional, but at least one parameter must be specified for this tag to be useful.
- A change to the timeout takes effect as soon as either timeout parameter is parsed and the timer is restarted at that point.
- It is strongly recommended that the inactivity and service timeout intervals are kept as short as possible. Setting the session inactivity limit to the maximum will keep sessions from automatically terminating when browser sessions are abandoned; this can result in a large number of orphaned BIS sessions that will not be cleaned up for over a week. It is better to set the inactivity timeout to 10 minutes and use a META REFRESH or a JavaScript timer to pull content from the BIS session every few minutes to keep the session active only while the browser window remains open.
- Setting the service timeout interval too high can also have detrimental effects if a service programs unexpectedly runs away. Such a program can use 100% of the available CPU, preventing any other programs from starting or running effectively. The default setting of 30 seconds will terminate any run-away program within this reasonable amount of time.
- The session scope determines if pages served from subdirectories of the session base directory are executed in new sessions. For example, if this page created the initial session:
  - `http://microfocus.com/xbis/apps/states/default.srf`

  and the application contains a link to this page, located in a more specific directory:
  - `http://microfocus.com/xbis/apps/states/texas/default.srf`
- If `SessionParms(Scope=All)` is in effect, the subordinate page will be served from the same session as the initial page. However, if `SessionParms(Scope=Isolate)` is in effect, a new session will be created for the subordinate page.
- For a description of the proper usage of the session base and session scope options, see *The Session Root Path and Session Scope* for more details.

The {{ServiceOpts}} Tag

This tag is a support tag for the `StartService` tag. It is for specifying options to the Visual COBOL runtime.

Note: The ServiceOpts tag is not supported by the IIS version of BIS for Visual COBOL.

```
ServiceOpts ( options )
```

where:

options	A space separated list of options to be passed to the service program. Individual options may be quoted, using either single or double quotes, with the opening quote type matching the closing quote type.

Notes

- The options are supplied to the service program in the order that they are specified in the `ServiceOpts` tag. Multiple `ServiceOpts` tags may be specified, with the options presented in the order that the tags are encountered in the SRF file. Only options specified up until the `StartService` tag are passed to the service program.
- If `ServiceOpts` tag is specified without the `( options )` parameter list, the set of options is emptied.
- The options are not saved in the session.

The {{ServiceArgs}} Tag

This tag is a support tag for the *StartService* tag. It is for specifying parameters to the Visual COBOL runtime. See the General Rules for the Format 1 ACCEPT statement in the Visual COBOL COBOL Language Reference, and the description of the command_line_linkage run-time tunable.

Note: The ServiceArgs tag is not supported by the IIS version of BIS for Visual COBOL. Use the SetEnv tag to set an environment variable, then, within the service program, use a Format 1 DISPLAY UPON ENVIRONMENT-NAME to set the name of the environment variable and a ACCEPT FROM ENVIRONMENT-VALUE to retrieve the value.

```
ServiceArgs ( arguments )
```

where:

arguments	A space separated list of parameters to be passed to the service program. Individual parameters may be quoted, using either single or double quotes, with the opening quote type matching the closing quote type.

Notes

- The arguments are supplied to the service program in the order that they are specified in the `ServiceArgs` tag. Multiple `ServiceArgs` tags may be specified, with the arguments presented in the order that the tags are encountered in the SRF file. Only arguments specified up until the `StartService` tag are passed to the service program.
- If the `ServiceArgs` tag is specified without the `( arguments )` parameter list, the set of arguments is emptied.
- The arguments are not saved in the session.

The {{ServiceLibs}} Tag

This tag is a support tag for the `StartService` tag. It is for specifying libraries to the service program.

Note: The ServiceLibs tag is not supported by BIS for Visual COBOL.

```
ServiceLibs ( libraries )
```

where:

libraries	A space separated list of libraries to be passed to the service program. Individual options may be quoted, using either single or double quotes, with the opening quote type matching the closing quote type.

Notes

- The libraries are loaded in the order that they are specified in the `ServiceLibs` tag. Multiple `ServiceLibs` tags may be specified, with the libraries presented in the order that the tags are encountered in the SRF file. Only libraries specified up until the tag are loaded.
- If `ServiceLibs` tag is specified without the `( libraries )` parameter list, the set of libraries is emptied.
- The libraries are not saved in the session.
- On UNIX, if a library does not begin with **lib**, it is automatically prepended. Furthermore, if the UNIX is a 64-bit operating system, and if 64-bit libraries have a suffix of 64 on this version of UNIX, 64 is appended to the library name. Finally, if library does not end with the proper extension for the UNIX operating system (for example `.so` or `.sl`), the proper extension is appended as well. These modifications to the library name are to allow the tag to be portable between Windows and UNIX.

The {{StartService}} Tag

This tag starts the execution of a service program. Options, parameters and libraries to be used by the service are specified by `ServiceOpts`, `ServiceArgs` and `ServiceLibs` tags.

```
StartService ( option program [parms] [, library1 [, library2] ...])
```

where:

option	The /CS_ANSI or /CS_OEM option can be used. If not specified, /CS_OEM is used by default, which results in unchanged behavior.
program	The name of the service program to run. If a relative path is specified, the path is relative to the directory that contains the `.srf` file. If no directory is specified, the RUNPATH is searched (see below).
parms	Optional Service Engine switches. Any text start at the first non-blank between *program* and the first comma or closing parenthesis is interpreted as a Service Engine option and is passed to the Service Engine without further interpretation. See the topic "Run-time Switches" in the Visual COBOL documentation. Parms is deprecated in favor of the `ServiceOpts` and `ServiceArgs` tags.
library	A comma-separated list of service libraries. If no extension is supplied under Windows, BIS will append `.dll` and on UNIX, BIS will `append.so`. This option is currently unsupported under Visual COBOL. *library* is deprecated in favor of the `ServiceLibs` tag.

BIS only allows one service program to be active in a session. Note the following:

- If no service program is currently running, the new service is started.
- If the specified service program is already running, this tag is ignored.
- If a service program is running, and *program* specifies a different service, the currently running service program is stopped (as if a `StopService` tag had been specified) and the new service program is started.

When a service program is started, BIS saves the name of the program. When another service program is started, BIS compares the new program name against the name of the program currently running. If there is an exact match (ignoring differences in letter case), the service is the same. If there is any difference, the new `StartService` tag refers to a different service and the currently running service program is stopped.

Once the service program is started, page rendering resumes. Rendering and the service program run in parallel.

Examples:

```
1. {{ StartService ( myapp.dll ) }}
2. {{ ServiceArgs  ( arg1 arg2 ) }}
   {{ StartService ( myapp ) }}
3. {{ ServiceOpts  ( +F +0 ) }}
   {{ StartService ( myapp.dll ) }}
4. {{ ServiceOpts  ( +T ) }}
   {{ SetEnv  ( COBCONFIG=myconfig.cfg ) }}
   {{ StartService ( myapp.dll ) }}
```

In the examples above, the `.dll`, and library files must be in a directory specified by , and `.CFG` files must be in the program_search_order.

1. Starts the program in file `myapp.dll`.
2. Attempts to start program myapp and pass it the arguments "arg1" and "arg2".
3. Starts the program in `myapp.dll` starting it with the +F switch (which checks all numeric items for valid numeric data) and sets COBOL switch 0 to ON.
4. Starts program `myapp.dll` starting it with the +T switch (which enables insertion of tab characters in all line sequential files). The `myconfig.cfg` run-time tunables file is processed when the service program is loaded.

The `StartService` tag follows all the regular Service Engine program loading rules. See the Visual COBOL documentation for a detailed description.

Accessing the REQUEST from the Service Program

In many cases, the service program requires access to the information transmitted in the HTTP request message. This information is passed in the BIS Request XML document that is made available by a call to `B_ReadRequest` within the service program.

Notes

- A given server response file can have multiple `StartService` tags. An additional `StartService` tag is ignored if it specifies a service that is already running. If it specifies a difference service, the service started by the previous tag is stopped before the new service is started.
- The `StartService` tag must precede any tags that depend on the service program being active. Such tags currently include XMLExchange.

The {{RunPath}} Tag

The `RunPath` tag is a deprecated tag and should not be used. Use the `SetEnv` tag instead to set the desired `COBPATH` environment variable.

This tag is used to modify the `RUNPATH` environment variable that is passed to the Service Engine. The BIS Service Engine uses the `RUNPATH` to locate service program files and libraries.

```
RunPath ( [ dir [,dir]… ] )
```

The {{SetEnv}} Tag

This tag is used to set a variable in the service program's environment.

```
SetEnv ( name[=value] )
```

Examples

```
{{ SetEnv ( printer=lpt1 ) }}
{{ SetEnv ( myfile=”c:\temp\scratchfile.tmp” ) }}
```

Notes

- The `SetEnv` tag affects only the Service Engine's environment and not the BIS Request Handler's environment. The `Value(variable,ENV)` tag will not retrieve variables set by this tag.
- Multiple `SetEnv` tags may be specified in a `.srf` file and are processed in the order in which they occur. Note that these tags must precede the `StartService` tag.
- On BIS/IIS, the scope of a `SetEnv` tag is the current request, not the current session. On BIS/Apache, the scope of the `SetEnv` tag is the current session.
- To unset an environment variable, omit the `=value`. Note that an unset variable is different from a variable that has a blank (or empty) value.

- All characters to the right of the equal sign up to the first space before the right-most parenthesis are stored as the value. If the value is quoted as in the example above, quotes will also be set in the environment.
- It is possible to use %VAR% sequences within the value portion of the `SetEnv` tag; in this case, %VAR% will be replaced with the contents of the VAR environment variable in the system environment (BIS/IIS) or the Apache server environment (BIS/Apache). The following restrictions apply:
  - On BIS/IIS, VAR must be set in the system environment. On BIS/Apache, VAR must be set in the Apache server environment or it must be a server variable. Variables in the BIS Service Engine's environment (like those set by previous `SetEnv` tags) are not considered. It is not possible to refer to the value set by one `SetEnv` in another `SetEnv` tag.
  - VAR is evaluated in the context of the request handler, not the service engine. While the service engine can be configured to run under an arbitrary user account, the request handler normally runs under a very restricted account that varies with the version of IIS and the version of Windows. %VAR% is therefore only useful for substituting system environment variables.

The {{XMLExchange}} Tag

This tag causes the web server to request text (typically XML, XHTML or HTML) from the currently running COBOL service program. The response text generated by the COBOL program replaces the `XMLExchange` tag in the output stream.

```
XMLExchange
XMLExchange ( OnExit=url )
```

The optional `OnExit` parameter determines the action that BIS takes if the service program is not active or terminates while the XMLExchange is being processed. It causes BIS to return an HTTP return code of 302 (`HTTP_REDIRECT_FOUND`) to the client. This causes the client to reissue the `GET` request for the specified URL.

Notes

- Do not use `OnExit` with SOAP requests. SOAP clients may not be able to interpret the 302 error that is returned.
- On BIS/IIS, the `OnExit` in the first `XMLExchange` tag following a `StartService` tag is ignored. This allows any service startup errors to be reported and corrected.

Recursive Tag Processing in {{XmlExchange}}

BIS recursively processes tags in the service program's response output, as if the response output was a stencil. Tag substitution occurs as the service output is written to the response page (replacing the `{{XmlExchange}}` tag), and substitution is performed in the context of the page that contains the `{{XmlExchange}}` tag.

This behavior allows the service program to dynamically generate tags, thereby using BIS tag substitution features in the HTML, XHTML or XML produced by the service program. For example, if the generated HTML contains URLs in links, the `{{Value}}` tag can be used to process those URLs in the context of the requested page, and make the URLs absolute, based on the URL of the original request. Another example: the service program can also change the content type or character set of the response by generating a `{{ContentType}}` tag.

The `{{FormActionTarget}}` tag discussed in the next section is a tag that is specifically intended to be included in generated `{{XmlExchange}}` output. Also note that any tag may be embedded in the output - even another `{{XmlExchange}}`.

The {{FormActionTarget}} Tag in {{XMLExchange}}

This tag is replaced by a URI referencing the current page and includes a query parameter that will be automatically checked by BIS to ensure proper sequencing of requests. BIS will check any requests to the

current session and will reject (displaying an error page) any request that does not contain the query parameter served by the `FormActionTarget` tag. By using this tag, the service program may assume that any requests that return control to the service are in the sequence expected by the program.

The `FormActionTarget` tag should normally only be used as the value of the *action* attribute of an HTML `<form>` element. In any case it must be used in such a way that the next expected request will be directed to the URI represented by the tag.

If a response page rendered by BIS does not contain the `FormActionTarget` tag, no sequence checking will be performed by BIS. The service program may, of course, perform its own checking using other means, such as hidden fields, if required.

The {{StopService}} Tag

This tag terminates the execution of the service program that is attached to the session.

```
StopService
```

Notes

- If the service program is not awaiting a request when this tag is rendered, the program must call `B_ReadRequest` within *ServiceTimeout* seconds. The call then returns with the `BIS-Fail-ProgramTerminated` return code. At that point, the program is granted an additional *ServiceTimeout* seconds to terminate.
- If the program is still running when either *ServiceTimeout* period expires, a termination signal is sent.
- Once the `StopService` tag is rendered, the service program is immediately disconnected from the session. For example, an `XMLExchange` tag immediately after a `StopService` tag is invalid and, if present, the `OnExit` parameter in that tag will be processed.
- The `StopService` tag may be immediately followed by a `StartService` tag. In this case, a new service program is started. Once the `StopService` tag is rendered, the service program is considered terminated even if it needs a few additional seconds to actually stop.
- This tag is ignored if there is no service program attached to this session.

The {{SessionComplete}} Tag

Indicates that the current session is complete and may be released. The session cookie will be deleted when the response for the current page is sent to the client.

```
SessionComplete
```

Notes

- If a BIS service program is currently active, this tag implicitly performs a `StopService` at the point this tag is rendered. See the description of the `StopService` tag for details about how the service program is informed the session is ending, and the sequence of events that transpire. Note, however, that the current or next call to `B_ReadRequest` returns the `BIS-Fail-SessionTerminated` result code instead of `BIS-Fail-ProgramTerminated`.
- This tag is most useful on a *goodbye* page, but is optional because sessions are automatically terminated after a period of inactivity. However, explicitly ending a session can be used to release system resources, or to force a new session to be started for the active client when the next page is requested.

The {{Value}} Tag

This tag looks up a value on the server and the tag is replaced with that value.

```
Value (variable|"variable" [, source] [,operations]…)
```

By default, *variable* is a server variable or a special variable (described below). However, options can direct that the value be obtained from the environment, the server configuration, a submitted form, a cookie, or a query parameter.

On BIS/IIS, if *variable* is enclosed in quotes, the variable name is treated as a literal string and is not resolved further unless one of the *source* options below is specified.

On BIS/Apache, if *variable* begins with a quote, it is treated as a literal and no *source* option is permitted.

Either single or double quotes may be used as delimiters, and a delimiting quote may be embedded in the string by specifying the quote twice: `"abc""def"` becomes `abc"def`.

The *source* option determines from where the *variable* value is obtained. If specified, the *source* must be the second parameter.

SERVER	Specifies that *variable* is a server variable. This is the default if none of the other sources below are specified and if the string is not quoted. Under BIS/Apache, ENV and SERVER are identical.
CONFIG	Specifies that *variable* is a special server configuration value. A list of CONFIG variables appears at the end of this section.
COOKIE	Specifies that *variable* is a cookie.
ENV	Specifies that *variable* is an environment variable instead of a server variable. Note that, on BIS/Apache, ENV and SERVER are identical. See *Setting Environment Variables* for information about setting and modifying environment variables on Windows. See *Configuring Apache* for information about setting environment variables on UNIX.
FORM	Specifies that *variable* is a <form> variable.
QUERYPARAM QP	Specifies that *variable* is a URL query parameter. QP is accepted as an alias for QUERYPARAM .

These *operations* modify the retrieved value and are applied from left to right and may be applied multiple times.

DEFAULT=*value*	Specifies a default value for *variable* if the *variable* is empty. This option has no effect unless the *variable* is empty at the point the DEFAULT operation is performed. • If the *variable* is empty when the end of the entire operations list is reached, the Value tag is simply removed from the output stream. • If DEFAULT is encountered and the *variable* is empty, the tag is replaced by *value*. If there are additional operations to the right of the DEFAULT operation (that is, GETDIR , TOUPPER, URLENCODE), these are performed on the new defaulted *value*.
GETDIR	Same as GETPATH (see below), except only the directory portion of the pathname is extracted. This is the part of the pathname that follows the scheme and hostname up to the last slash in the pathname. Note that if *variable* is a

	pathname that contains a drive letter, the drive letter is also returned. The extracted pathname never ends in a slash.
GETQUERYSTRING	If *variable* is a URL, returns the query string. If not present or if *variable* is a pathname, an empty string is returned.
GETHOST	If *variable* is a URL, extracts the hostname. If *variable* is a pathname, or no host name is present, an empty string is returned.
GETNAME	Same as GETPATH , except only the filename portion of the pathname is extracted. This is the part of the pathname that follows the last slash but excludes the #fragment , ?querystring, and ;parameters.
GETPATH	If *variable* is a URL, extracts the path portion of the URL. This is the portion of the URL that excludes the scheme, the hostname, and the query string. If variable is a pathname, it is unchanged.
GETSCHEME	If *variable* is a URL, extracts the scheme. This will normally be http or https without the terminating colon or slashes. If *variable* is a pathname or a URL without a scheme, an empty string is returned.
SUBSTITUTE= *pattern/replacement/* SUB=*/pattern/replacement/*	Allows you to substitute all occurrences of *pattern* in the value with a replacement pattern. The operation is performed on the current value after all transforms to the left have been performed. Processing continues with the modified value. SUB is accepted in place of SUBSTITUTE for brevity. Both *pattern* and *replacement* are regular expressions. (For more information, see *Regular Expression Syntax*).
TOUPPER	Converts the value to all upper-case characters. Equivalent to SUBSTITUTE="/.*/\U&/".
TOLOWER	Converts the value to all lower-case characters. Equivalent to SUBSTITUTE="/.*/\L&/".
URLDECODE	Decodes a string that has been URL-encoded. This is primarily useful when retrieving a server variable.
URLENCODE	Encodes a string for reliable HTTP transmission from the web server to a client as a URL. For example, This is a <Test String>. will be encoded as: This%20is%20a%20%3cTest%20String%3e.
HTMLDECODE	Decodes a string that has been HTML-encoded. This is primarily useful when retrieving a server variable.
HTMLENCODE	Encodes a string for reliable HTTP transmission from the web server to a client as HTML. For example, This is a <Test String>. will be encoded as: This is a <Test String>.
MAKEABS	Assumes that the string is a relative URL, and makes the URL absolute, using the location of the stencil that was requested by the client as the base URL (see REQUEST_URL in *Configuration Variables* for details). If the string is not a URL, it is not altered. If the input string is an absolute URL, it is cleaned up (that is, redundancies

	such as `dir/../` are removed) but is otherwise unchanged. See RFC 3986 for details about how relative URLs are resolved by this operation.

Processing stops when the following option is encountered and the tag always renders as an empty string.

`MATCH=regexp`	Applies the regular expression against the current value and returns true if it matches and false if it does not match but does not return any text for rendering. This allows `Value` to be used in `If` tags. See *Regular Expression Syntax*.

For example, the tag:

```
{{ VALUE (HTTP_URL, GETDIR, TOLOWER, URLENCODE) }}
```

is replaced by the directory that contains the page that is currently being served. The name of the directory is converted to lowercase and the directory name is URL-encoded (for example, recommended if the value will be substituted into an `HREF` attribute). `HTTP_URL` is a server variable, but it is not necessary to specify the `SERVER` source parameter because this is the default.

Notes

- The `Value` tag can be referenced in an `If` tag if the `MATCH` operation is used, but cannot be nested within any other tags. It can, however, appear anywhere else in the HTML as long as it follows the `Handler` tag. This tag can therefore be used to provide content for any HTML element.
- When used in an `If` tag without the `MATCH` option, the condition is `TRUE` if `Value` evaluates to a non-empty string; otherwise, `FALSE`.
- Regular expressions must be delimited. The first nonblank character after the = is the delimiter for the regular expression. The expression begins at the character following the delimiter and extends up to, but not including the next occurrence of that character.

  Single or double quotes are common delimiters, but the delimiter may be any character. Examples:

  ```
  1. {{ VALUE (QUERY_STRING, SERVER, MATCH=”?userid=fred\s”) }}
  2. {{ VALUE (QUERY_STRING, SERVER, MATCH=/?userid=”fred\s”/) }}
  ```

  (Note that `QUERY_STRING` is a server variable that contains the query string part of the URL.)

  The second regular expression includes quotes, so a delimiter (`/`) was chosen that does not occur in the expression.

  Another way to accomplish the above is to use the `QUERYPARAM` source option:

  ```
  {{ VALUE(userid, QUERYPARAM, MATCH=”fred\s”, URLENCODE) }}
  ```
- Commas cannot occur inside delimited or quoted strings because commas always separate parameters. If a comma is required, use `"%2c"` and `URLDECODE` the string to convert the `"%2c"` to a comma.

Configuration Variables

In addition to server variables and environment variables, some special variables are supported. These variables may not be implemented on all platforms.

`HOSTSERVER`	Returns `IIS` or `Apache`. Note that under IIS, the `SERVER_SOFTWARE` server variable can be used to retrieve the version number. However, this server variable may be undefined under Apache.

MAXTHREADS IIS	Resolves to the number of threads configured in the BIS thread pool. This is the number of threads that are available for requests. Under Apache, this is undefined (use DEFAULT=1 if portability is desired).
REQUEST_URL	Retrieves the completely qualified URL of the stencil (.srf) file that was requested by the client. This includes the scheme, hostname, port number (if non-standard), path, and parameters, query string, and fragment (if specified).
REQUEST_BASE_URL	Retrieves the completely qualified base URL of the current stencil (.srf) file that was requested by the client. This includes the scheme, hostname, port number (if non-standard), and the path (which always ends in a slash). The base URL is defined as the directory that contains the stencil that was requested by the client.
STARTSERVICE	Returns the entire argument list of the currently active StartService tag, including commas. If there is no active service program, the value is considered undefined and may be overridden with the DEFAULT operation.

SCHEME	Returns the scheme that was used to request the current page: currently returns either http or https. Note that the :// delimiter that follows the scheme is not included. This is useful for constructing URLs:
	`<a href="{{ Value(SCHEME, CONFIG) }}://`
	`{{ Value(HTTP_HOST,URLENCODE) }}`
	`{{ Value(HTTP_URL,GETDIR,URLDECODE,URLENCODE)`
	`}}/default.srf">`
	(Note that the above must be on a single line, or spaces will be inserted.) Under BIS/IIS, the scheme is derived from the SERVER_PORT_SECURE server variable, where a value of 0 indicates http and nonzero indicates https.

SERVICENAME	Retrieves the name of the currently active service program. If there is no active service program, the value is considered undefined and may be overridden with the DEFAULT operation.
VERSION	Retrieves BIS version information. The format of the version number is aa.bb.cc.yyyy/mm/dd, where aa.bb.cc indicates the numeric major/minor/patch level version, and yyyy/mm/dd is the build date.

The {{Trace}} Tag

Enables or disables trace logging for the current session.

```
Trace ( options )
```

The options in the table below control the internal accumulation of trace information on UNIX. Windows always accumulates trace information and these options are ignored.

START	START causes BIS to begin accumulating trace output. If tracing has been started, START has no effect.

STOP	STOP causes BIS to stop accumulating trace output. If tracing has not been started, STOP has no effect.
OFF	Turns tracing off. Equivalent to STOP,NOPAGE,NOFILE,NOTAG,NOEXCHFILES,NOQUERYPARAM,NOIP.

The options in the table below determine where the TRACE output is emitted. They are independent of each other. The underlined options are the defaults.

PAGE	Indicates that the trace is emitted at the end of the page.
NOPAGE	Disables end of page trace output.
FILE	Indicates that the trace is written to a file in directory indicated by the DIR option.
NOFILE	Disables trace output to the file.
TAG	Enables the TraceDump tag and allows it to write trace output is written when it is rendered.
NOTAG	Causes TraceDump tags to be ignored.
EXCHFILES	Enable saving a copy of the XML Exchange request/response files for each session in the trace directory.
NOEXCHFILES	Disables the tracing of XML Exchange request/response files.

If the FILE option is in effect, these options determine how the TRACE output is written to a file.

DIR=*dir*	*dir* specifies the directory that will receive trace output if FILE is in effect. If no *dir* is specified, this option has the same effect as NODIR. If a relative directory is specified for *dir*, output is written into a directory relative (on BIS/IIS) to the Windows temporary directory or (on BIS/Apache) to the /tmp directory. If an absolute path is specified for *dir*, output is written into that directory. On BIS/IIS, this directory must exist or the trace file will not be written. On BIS/Apache, the specified directory will be created if it does not exist.
NODIR	Disables the trace directory specified by DIR. If file output is enabled with either FILE or EXCHFILES then all trace output is written (on BIS/IIS) into the Windows temporary directory, or (on BIS/Apache) into the directory specified by the BISTraceDirectory Apache configuration directive.

The options below allow tracing to be controlled using a query parameter or a cookie:

QUERYPARAM=*value* QP=*value*	QUERYPARAM and QP are synonymous and designate a URL query parameter whose value can be used to dynamically specify the options above. See the section *The Trace Query Parameter* for more information. Since the ability to dynamically configure the trace system is a potential security issue, the QUERYPARAM option allows you to specify your own query parameter name, rather than BIS supplying a standard one. This will make it harder for an unscrupulous person to obtain control of the tracing, but not impossible, so it is strongly suggested that the QUERYPARAM option not be left in the Trace tag of stencils files in production systems.

NOQUERYPARAM	Disable the query parameter set by QUERYPARAM or QP.
NOQP	
IP=xx.xx.xx.xx [-x.xx.xx.xx] IP=*ipv6addr* [- *ipv6addr*]	IP allows trace output to be restricted to requests originating at one or more IP addresses. If an IP restriction is in effect, trace output is restricted exclusively to requests from those particular IP addresses. A comma-separated list of IP addresses or ranges may be specified. The list of IP restrictors is processed from left to right. Note that specifying 127.0.0.1 will allow access from a web browser running on the host's console. In this case, access the pages using localhost as the name of the host. IPv6 addresses may be specified instead of IPv4. For example, ::1 specifies the IPv6 loopback address, and on almost all Windows versions now, localhost resolves to this IPv6 address; while on older versions of Windows, localhost resolves to 127.0.0.1. IPv6 and IPv4 addresses may be mixed in a single IP statement, but not in a single range. If either an IPv4 or IPv6 loopback address is specified (that is, 127.*.*.* or ::1), the setting applies to both IPv4 and IPv6 loopback addresses.
NOIP	Disables the restriction of IP addresses.

Notes

- The default trace state is OFF. Note that if Trace(Start) is specified, trace accumulation begins/continues but trace information is not output until one or more output destinations (that is, PAGE, FILE, TAG, EXCHFILES) are specified.
- The trace mode is part of the session and is *sticky*. This means that the trace setting persists in the session until it is changed by either another trace tag or a query parameter (if enabled). So if you have more than one page in your application, the trace tag is required only on your initial page.
- Only .srf files may be traced. If you follow a link to an .htm or .asp page, those pages will not be traced. If those pages link back to a .srf file in this application's virtual directory, then tracing will once again resume as long as the session is still active.
- Be cautious when enabling tracing in a way that exposes the trace information to site visitors. Trace information will reveal some information about your system that may be useful to intruders. The QUERYPARAM is configurable to help secure your web by allowing tracing to be turned on and off using a keyword that is not easily guessed by intruders.

Examples

```
{{ trace(page, file, notag, dir=bistrace, ip=127.0.0.1) }}
```

This Trace tag directs that trace output will be appended to every HTML page, and will also be written to the trace file in a directory named bistrace-note that, on Windows, this directory is relative to the Windows temporary directory, and must exist. The notag option causes TraceDump tags in the stencil file to be ignored and page trace output is only performed if the request originates on the server running BIS via the localhost alias (always 127.0.0.1).

Note that specifying 127.0.0.1 (or any IPv4 loopback address) also enables tracing from ::1 (the IPv6 loopback address), and vice-versa.

The Trace Query Parameter

If the QUERYPARAM option was specified in the Trace tag, it defines a query parameter that may be specified on the URL of a request by the client. If that query parameter is present, then its value will be parsed for trace options to use to configure the tracing. These options are, as with those specified by the Trace tag itself, persistent and will stay with the session until it completes, or a Trace tag or a trace query parameter alters it further.

For example, if the .srf contains the following tag:

```
{{ trace (QueryParam=ClientQuery) }}
```

Tracing will not occur for normal requests. However, the following request is made:

```
http://localhost/xbisvcob/samples/mf/verify?ClientQuery=page
```

Page tracing will be turned on for the session, and tracing will be appended to the output for every subsequent request. See *The BIS_TRACE_SUFFIX Environment Variable* for another example.

Trace options set using the trace query parameter have the highest priority. Note that, for security, the query parameter cannot be used to set or clear IP restrictions or set the trace output directory.

The BIS_TRACE_SUFFIX Environment Variable

On BIS/IIS, the BIS_TRACE_SUFFIX environment variable and, on BIS/Apache, the BISTraceSuffix configuration parameter allows trace parameters to be injected into every trace statement. While this requires administrative access to the web server, this is useful for globally providing specific clients access to trace information.

For example, if your trace statements look like this:

```
{{ trace(page, noip) }}
```

and you wish to view trace data from the machine at 192.168.3.54, and control such tracing with the MySecretTrace query parameter, place this into the server environment:

```
BIS_TRACE_SUFFIX=ip=192.168.3.54,queryparam=MySecretTrace
```

- This will effectively append these parameters to every Trace tag executed on the server without requiring the actual .srf file to be edited. Note that the .srf files must contain a Trace tag for this feature to take effect.
- See *Setting Environment Variables* for information about setting and modifying environment variables on Windows. See *Configuring Apache* for information about setting Apache configuration parameters.

The {{TraceDump}} Tag

This tag directs BIS to output the contents of the trace buffer.

```
TraceDump
```

Notes

- This tag is ignored (that is, removed from the output) if tracing is not being performed.
- Because trace information is accumulated as the page is rendered, it is most useful for the TraceDump tag to be specified near the end of the page.
- If this tag is omitted and page tracing is enabled, BIS/IIS appends trace output to the end of the response (that is, after the </html> tag).

The {{Debug}} Tag

IIS

This tag enables or disables service engine debugging for the current session.

Note: The Debug tag is not supported by BIS for Visual COBOL.

The Debug tag performs the following functions:

- Determines if debugging will be enabled or disabled
- Optionally specifies the addresses of the clients that are allowed to debug this session.

The syntax of the `Debug` tag:

```
{{Debug (option [,option]...}}
```

Options

`ON`	Enables service program debugging in this session. `ON` allows service programs that are subsequently created in the current session to be debugged.
`OFF`	Disables service program debugging in this session. `OFF` prevents service programs created in this session from being debugged and clears the list of ID strings and IP restrictions.
`QUERYPARAM` `QP`	`QUERYPARAM` and `QP` are synonymous and contains the name of a query parameter that can be used to dynamically specify debug parameters. Debug options set with `QUERYPARAM` will override those set in the `DEBUG` tag. However, for security reasons, the query parameter can only be used to turn debugging on and off.

`IP=n.n.n.n [-n.n.n.n]`
`IP=n.n.n.n /n`
`IP=ipv6 [- ipv6 ]`
`IP=ANY|ALL`
	Allows debugging to be restricted to requests originating from one or more IP address. If an IP address restriction is in effect, debug requests will only be accepted from clients that have IP addresses assigned within the restricted range. A space-separated list of IP addresses or ranges may be specified. The list of IP restrictors is processed from left to right.
	If `ANY` or `ALL` is specified, requests from any IP may be debugged.
	Note that specifying either `127.0.0.1` or `::1` will allow access from a web browser running on the host's console. In this case, access the pages using localhost, `127.0.0.1`, or `[::1]` as the name of the host.
	If the `OFF` action (above) is specified, the IP restriction list is cleared (same as `IP=ANY`), but any IP restrictions specified in the same or later tags will be processed and stored.
	If either an IPv4 or IPv6 loopback address is specified (that is, 127.*.*.* or ::1), the setting applies to both IPv4 and IPv6 loopback addresses.

`NOIP`	Same as `IP=ANY`.

IIS Debugging Notes

- To enable debugging and clear all IDs and IP restrictions that may have been previously set for the current session, specify `{{ Debug(OFF,ON) }}`.
- When a service is started and the debugging is enabled, the debugger will start running on the host machine's console. For this reason, it is desirable to restrict debugging to local host.
- The `Debug` tag is always removed from the rendered page. If this is the last tag on the line, a newline is output unless this tag is immediately followed by a comment tag. See the *Notes* section of *Comment Tags*.
- The `Debug` tag is ignored unless debugging is enabled in the BIS web server's configuration.

  Tip: To configure debugging for all pages, place the list of authorized users and IPs into the `web.config` file and add this tag to all of your `.srf` files:

  ```
  {{ Debug( ON,ID={%DEBUG_USERS, CONFIG%},IP={%DEBUG_IPS,CONFIG%} ) }}
  {{ StartService( MAINPROGRAM ) }}
  {{ Debug( OFF ) }}
  ```

  Then, create or edit the `web.config` file in the directory containing the .srf file (or any parent) and add the variables below:

  ```
  <app.config>
     <configuration>
          ...
        <configSections>
           <sectionGroup name="BIS">
              <sectionGroup name="Config">
                 <section name="Variables"
     type="System.Configuration.NameValueSectionHandler,System"/>
              </sectionGroup>
           </sectionGroup>
        </configSections>
          ...
        <BIS>
           <PreRender>
                 <add tag=”Debug(ON,ID=Users,IP=ANY” />
           </PreRender>
           <Config>
              <Variables>
                 <add key="DEBUG_USERS" value="[spaced-list]"/>
                 <add key="DEBUG_IPS" value="[spaced-list]"/>
                   ...
              </Variables>
           </Config>
        </BIS>
          ...
     </configuration>
  </app.config>
  ```

- `Debug` tags are processed in the order that they are encountered during rendering. This means that, in this example:

  ```
  {{ Debug( ON, IP=127.0.0.1 ) }}
  {{ Debug( OFF ) }}
  {{ StartService( MAINPROGRAM ) }}
  ```

  Debugging of `MAINPROGRAM` will be disabled. However, in this case:

  ```
  {{ Debug( ON, IP=127.0.0.1 ) }}
  {{ StartService( MAINPROGRAM ) }}
  {{ Debug( OFF ) }}
  ```

  Debugging of `MAINPROGRAM` will be. Subsequent programs will not be debugged unless an intervening `{{ Debug(ON) }}` is rendered and a new ID and optional IP restriction is set.

Control Flow Tags

Control flow tags determine how Business Information Server processes a particular server response (`.srf`) file. These tags are similar to the "C" `#if/#else/#endif` and `#include` preprocessor macros.

There are two control flow tags:

- `If`/`Else`/`Endif` that may be used to prevent a section of the file from being rendered.
- `While`/ `EndWhile` that may be used to repeat a section of HTML code.
- An `Include` tag that can be used to embed one stencil or into another.

The {{If}} / {{Else}} / {{EndIf}} Tags

These tags can be used to conditionally prevent sections of the stencil file from being rendered.

```
{{ if Value(parameters) }}
     if-content
{{ else }}
     else-content
{{ endif }}
```

Notes

- The `Value` parameters list has the same syntax as the parameters list for the `Value` tag: see *The {{Value}} Tag*. However, note that the parameters list must result in a `TRUE`/`FALSE` result, and must therefore contain a `MATCH` operation.
- The definition of content includes both HTML/XML and replacement tags.
- Any HTML/XML code in a skipped section is ignored and is not transmitted to the user agent. Server response file tags in a skipped section are ignored and are not evaluated.
- No special flow layout is implied by this tag: the `If`, `Else`, and `EndIf` tags can be on one line, or can span multiple lines.
- Blocks may be nested but must be completely nested. It is not permissible to place a `While` tag in an `If` block and have the `EndWhile` tag in a different block.
- To render on an inverted condition, just omit the if-content: `{{ if tag }}{{ else }}` *content* `{{ endif }}`.
- If the `If` / `Else` / `EndIf` tag is the last tag on a line, a newline will be added. If this tag is the only tag on the line, a blank line will be output. To avoid this, place a comment tag at the end of the line. For example, `{{ EndIf }}{{//}}`.

The {{While}} / {{EndWhile}} Tags

This tag can be used to omit or duplicate a section of HTML code.

```
{{ while Value(parameters) }}
     content
{{ endwhile }}
```

Notes

- The `Value` parameters list has the same syntax as the parameters list for the `Value` tag: see *The {{Value}} Tag*. However, note that the parameters list must result in a `TRUE`/`FALSE` result, and must therefore contain a `MATCH` operation.
- The definition of *content* includes both HTML/XML and replacement tags.
- No special flow layout is implied by this tag: the `While` and `EndWhile` tags can be on one line, or can span multiple lines. These blocks can also be nested.

- A `While` block must be completely enclosed within another `While` block, or the true or false section of an If block. It is not permissible to use an If block to conditionally render an `EndWhile` tag unless the `While` tag is in the same block.
- If the `While` / `EndWhile` tag is the last tag on a line, a newline will be added. If this tag is the only tag on the line, a blank line will be output. To avoid this, place a comment tag at the end of the line. For example, `{{ EndWhile }}{{//}}`.

The {{ Include }} Tag

This tag is replaced by the contents of the specified file.

```
include filepath
```

Where *`filepath`* is the path to a target file whose contents, when rendered, will replace the `include` tag. You may specify an absolute path or a path relative to the physical location of the `.srf` file that contains the `Include` tag.

If the target file is a server response file, and contains a handler tag, the target file is independently processed (rendered) in its own context, and this is recursively repeated for any tags in the target file. When rendering is complete, the rendered output replaces the `Include` tag.

If the target server response file does not contain a `Handler` tag, it is treated as a text file and replaces the `Include` tag without further processing. Any unresolved tags will not be processed, but will remain in the final response

Notes

- Relative pathnames in *`filepath`* are interpreted as relative to the location of the `.srf` file that contains the `include` tag. This is also true for any additional nested `Include` tags: the path is always relative to the server response file that is being processed.
- If an included `.srf` file contains a `StartService` tag, the service program's working directory is the directory that contains the `.srf` file that rendered the tag.
- The included file does not need to be a `.srf` file. For example, an `.html` file, a .`css` (cascading style sheet) file, or a `.js` (JavaScript) file can also be included, and in this case, the `Include` tag is simply replaced by the content of the specified file.
- On BIS/IIS, an `include` tag can appear anywhere in a `.srf` file-even before the `handler` tag.
- If an `Include` tag is the last tag on a line, it will be followed by a newline unless immediately followed by a comment tag.

{{//}} Comment Tags

This tag is ignored and is simply removed from the output. Comment tags differ from HTML comments, which remain in the output and can be viewed with the browser's **View > Source** command.

There are two ways to specify a BIS comment:

```
{{ // comment }}
{{ !-- comment }}
```

Notes

- A comment tag can appear anywhere in a server response file-even before the `Handler` tag.
- If a comment tag is immediately followed by the end-of-line character, the newline character is removed with the comment tag. This is useful when placing tags into a file where white space is significant. For example, a server response file could be coded like this:

```
{{//There must be no whitespace rendered before the exchange tag, }}
{{// hence the newline-eating comment tags }}
{{ Handler * }}
```

```
{{ Trace(start,queryparam=trace,ip=127.0.0.1) }}

{{ SetEnv(COBCONFIG=../common/cblconfig.txt) }}
{{ StartService(webappsample2.dll) }}
{{ XMLExchange(OnExit="gotit.srf") }}
```

Here, the `Handler`, `Trace`, `SetEnv`, and `StartService` tags are completely removed from the output, while the `XMLExchange` tag is replaced by the XML produced by the COBOL program. However, the new line character that follows each of these tags would remain in the output, resulting in four blank lines before the start of the XML produced by the `XMLExchange` tag.

To avoid this in this sample, the non-comment `Handler`, `SetEnv`, and `StartService` tags are followed by empty comments, which suppress the newline characters. The `XMLExchange` tag is not followed by a newline-consuming comment because a newline is desirable before the end of the file and, in this case, the emitted XML does not contain any newline characters.

```
{{//There must be no whitespace rendered before the exchange tag, }}
{{// hence the newline-eating comment tags }}
{{ Handler * }}{{//}}
{{ Trace(start,queryparam=trace,ip=127.0.0.1) }}{{//}}

{{ SetEnv(COBCONFIG=../common/cblconfig.txt) }}{{//}}
{{ StartService(webappsample2.dll) }}{{//}}
{{ XMLExchange(OnExit="gotit.srf") }}
```

Here, the comment tags and the `Handler`, `Trace`, `SetEnv`, and `StartService` tags are completely removed from the output, while the `XMLExchange` tag is replaced by the XML produced by the COBOL program.

Service Programs

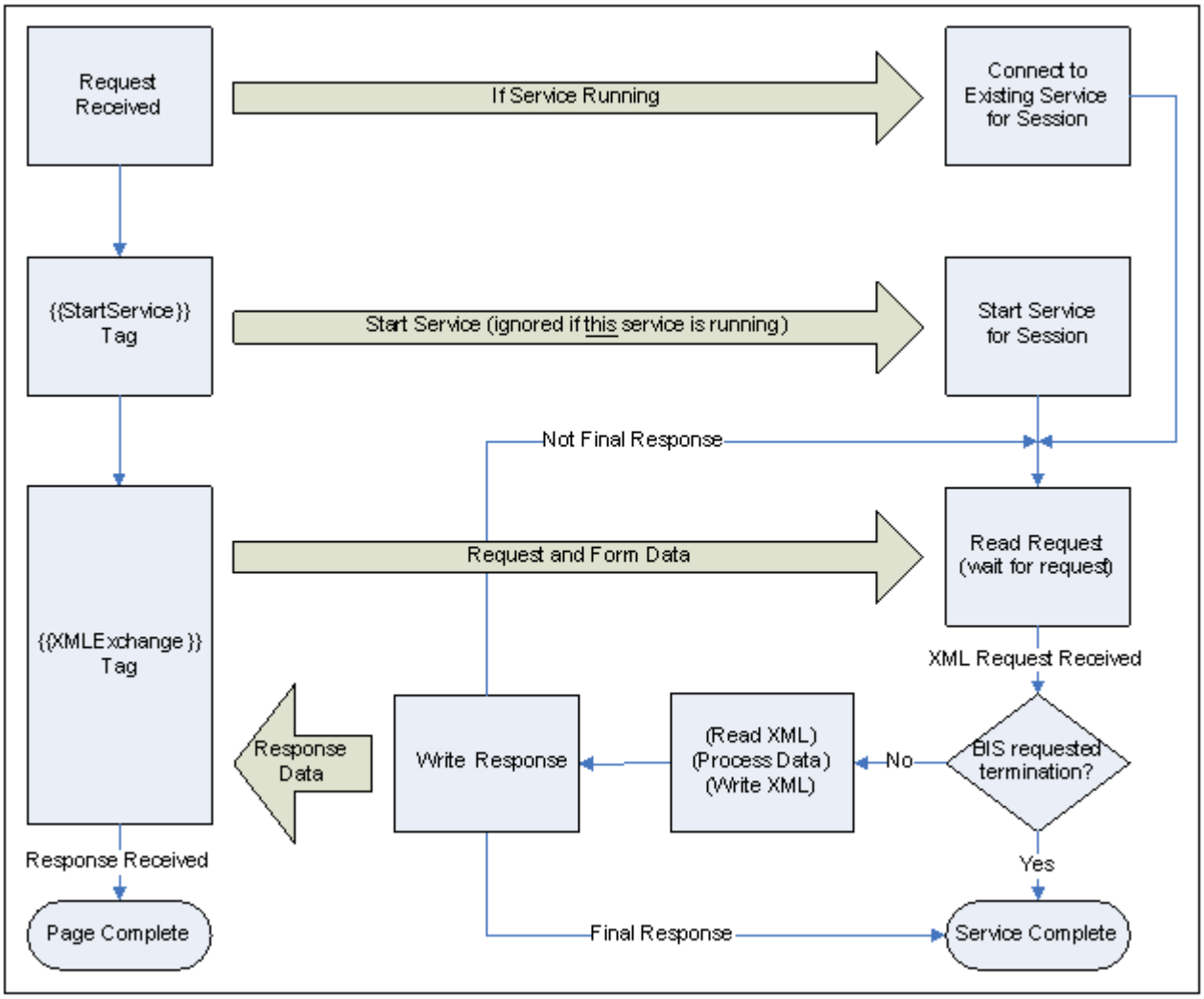
Introduction

The *Service Engine* is the BIS component that starts and runs service programs in response to requests. Currently, all BIS service programs are COBOL programs.

The Service Engine is started when a BIS `StartService` tag is rendered, and runs asynchronously from the BIS web components. BIS and the Service Engine synchronize when:

1. BIS renders an `XMLExchange` tag, and
2. The Service Engine calls `B_ReadRequest`.

The simplified flow of control is depicted below.



The BIS Request Handler and the BIS Service Engine synchronize when the Request Handler renders an `XMLExchange` tag and the Service Engine calls `B_ReadRequest`. Ideally, the Service Engine will be waiting at a synchronization point when the BIS Request Handler is ready to provide a request. To avoid deadlocks, once BIS begins to process the `XMLExchange` tag:

- The service program must call the `B_ReadRequest` function within *`ServiceTimeout`* seconds.
- Alternatively, the program may request additional time by calling `B_SetServiceTimeout` using `0` to reset the timer.

Once the Service Engine has accepted the request, it is granted a new *`ServiceTimeout`* interval to read the XML request, compute the response, write the XML response, and call `B_WriteResponse`. Alternatively, the service program can terminate, which causes the BIS Request Handler to redirect if an `OnExit` parameter was specified in the `XMLExchange` tag. If the response cannot be provided within this interval, the service program must request more time as described above.

When the BIS Request Handler receives the response, it is placed into the page output stream and processing continues. At this point, the Service Engine may:

- Wait for the next request for the current session by calling `B_ReadRequest`.
- Terminate (for example, with `GOBACK`.)

If the service program does neither of these, but instead keeps running, the Service Engine eventually terminates it with a Service Timeout.

Service Program Lifetime

A service program is started when BIS processes a `StartService` tag on a `.srf` page. A service program is considered to be finished when:

- The program terminates by executing a `GOBACK` (or equivalent).
- The program responds to a request by calling `B_WriteResponse` with an `end program` or `end program and session` disposition parameter (described in detail in *BIS Return Codes*).
- A `StopService` tag is rendered. The service program is disconnected from the session, so a subsequent `StartService` can be processed on the same page.
- A `SessionComplete` tag is rendered. The service program and session both end when the page is complete. Note that a `StopService` can also be specified if the service program must stop immediately.
- The number of seconds specified in the *`InactivityTimeout`* pass without a request. Both the service program and the session are terminated.
- An `XMLExchange` tag is rendered and the number of seconds specified in the *`ServiceTimeout`* interval pass without a response from the service program. If a service program needs a longer amount of time to complete processing, it should lengthen the *`ServiceTimeout`* interval by calling `B_SetServiceTimeout()`, or call this function with a parameter of zero to reset the timer.

The following general rules apply to service programs:

- A given BIS session may have only one active service program at any time.
- When a service program enters the termination state, it is immediately disconnected from the session but is given 30 seconds to clean up and perform a `GOBACK`. If the program is still running when the timer expires, BIS requests that the program stop at the next statement boundary and the service program is granted another 30 seconds to terminate. If, at the end of the allotted time the program has still not terminated, the process is forcibly terminated and unloaded from memory.

A new service program may be started as soon as the current service program is disconnected from the session. In other words, `{{StopService}} {{StartService(…)}}` is allowed.

ACCEPT and DISPLAY Statements

Because the service program does not have access to the console or the Windows desktop, the behavior of `ACCEPT` and `DISPLAYS` that involve screen or terminal I/O is undefined and must be avoided.

To write trace information into the BIS trace output, use `B_Trace`.

Windows Message Boxes and Dialog Boxes

Because the service program does not have access to the Windows desktop, it is not appropriate to display a message box or a dialog box. If the service program did attempt to interact with the user in this way, it will suspend waiting for a response that cannot ever come.

The XML Exchange File

The Service Engine is started with a special parameter that specifies the name of the file that will be used for all XML exchange operations. The BIS Request Handler takes the current request, encodes it using XML, and places the request into this file when the service program calls `B_ReadRequest`. If desired, the `B_ReadRequest` can also pass the XML Exchange information strictly via memory.

Important: The file is not created until `B_ReadRequest` is called.

BIS places the fully qualified name of this file into the BIS_FILENAME environment variable when the Service Engine is started. The filename, therefore, is accessible to the COBOL program via the environment:

```
01 BIS-Exchange-File-Info.
   05 BIS-Exchange-File-Result    PIC 9 BINARY.
   05 BIS-Exchange-File-Name.
      10 FILLER                  PIC X OCCURS 200 TIMES.


DISPLAY "BIS_FILENAME" UPON ENVIRONMENT-NAME.
ACCEPT BIS-Exchange-File-Info FROM ENVIRONMENT-VALUE.
```

On BIS/IIS, the value of this variable is the fully qualified pathname of the file and the filename has this form:

```
XMLExchange-hhhhhhhh-hhhh-hhhh-hhhh-hhhhhhhhhhhh.xml
```

The file is created in the Windows TEMP directory. The *h* characters are replaced by hexadecimal digits, and the name is guaranteed to be globally unique.

On BIS/Apache, the value of this variable is the fully qualified pathname of the file and has this form:

```
bisiiiiiiiiiiiiiiiiiiii-ssss.xml
```

The file is created in the directory indicated by the Service Engine's TempDir configuration keyword. The *i* characters are replaced by the session's identifier and the *s* characters are replaced by a decimal number representing the number of the service within the session.

Notes

- You do not provide this environment variable. BIS sets the environment variable when a service program is started and creates the file when B_ReadRequest is called.
- A separate file is created for each service program, and the same file is used by
  - B_ReadRequest to receive requests from BIS.
  - B_WriteResponse to transmit responses to BIS.
- While the filename is already known when the service engine is started, the file itself is not created until B_ReadRequest is called for the first time by the service program.

BIS Return Codes

Here are the return codes for the B_ functions. These codes are defined in the bisdef.cpy COPY file provided in the samples/common directory. Note that the severity of an error condition increases with the value of the return code

00-09	Success! For B_ReadRequest, the request data is available in the XML exchange file. For B_WriteResponse, the response was accepted by the Request Handler. • 00 - BIS-Success: The data transfer succeeded and the XML file contains the result of the operation.
10-19	A non-fatal event occurred, and recovery is possible. While no such conditions currently exist, these codes are reserved for future use.
20-29	A failure occurred but the program should be able to recover. The states of the service program and request handler have not been affected by this operation.

	• `20 - BIS-Warn-RequestTimeExpired`: A timeout parameter was specified on the `B_ReadRequest` call and the timeout expired. To avoid a potential race condition, the service program should not terminate when this occurs-instead, it can do some work and then reissue the request. • `21 -BIS-Warn-RequestOutstanding`: A request has already been received by `B_ReadRequest` and the service program has not responded. This code is only returned by BIS/Apache; BIS/IIS recreates the exchange file and returns `BIS_Success` each time that `B_ReadRequest` is called for the same request. • `22 - BIS-Warn-ResponseUnexpected`: The service program called `B_WriteResponse` without a pending request. • `23 - BIS-Warn-CallNotImplemented`: A function was called that is not implemented in this version of BIS.
`30-49`	A failure occurred. The service program may attempt to recover, correct the problem and retry the operation. The state of the service program and request handler have not been affected by this operation. • `30 -BIS-Fail-FileOpen`: BIS could not open the XML Exchange file. • `31 - BIS-Fail-FileRead`: BIS could not read the XML Exchange file. • `32 - BIS-Fail-FileWrite`: • `33 - BIS-Fail-FileClose`: BIS could not close the XML Exchange file. • `34 - BIS-Fail-FileSize`: The XML Exchange file size is too large to load into memory. • `39 - BIS-Fail-FileTraceFileIO`: BIS could not open or write the trace file. • `49 - BIS-Fail-FileFormat`: The XML Exchange file format is invalid.
`50-79`	A failure or possible planned session/service expiration event occurred and the program cannot continue. The XML exchange file was not updated and the program should terminate as soon as possible or BIS will terminate the program after the service timeout interval expires. • `50 - BIS-Fail-SessionAbandoned`: The session timed out. • `51 - BIS-Fail-SessionComplete`: The user logged out or ended the session. Note that this does not necessarily indicate a failure-a `SessionComplete` tag may have been processed. • `52 - BIS-Fail-ServiceComplete`: The user logged out or ended the session. Note that this does not necessarily indicate a failure-a `StopService` tag may have been processed.

`80-99`	A serious error occurred. The XML exchange file was not updated and the program must terminate as soon as possible or BIS will terminate the program after the service timeout interval expires. • `80 - BIS-Fail-ServerUnavailable`: The service program is not running in the BIS server environment. • `81 - BIS-Fail-ServerUnspecified`: An unspecified error occurred while the service program was communicating with the BIS Request Handler. • `88 - BIS-Fail-ServerInternalError`: An internal error occurred while the service program was communicating with the BIS Request Handler. • `89 - BIS-Fail-ServerMemoryManagement`: A memory management failure occurred in the BIS service program. • `90 - BIS-Fail-ServerBadMessage`: An internal error occurred while the service program was communicating with the BIS Request Handler. • `91 - BIS-Fail-ServerBadLength`: An internal error occurred while the service program was communicating with the BIS Request Handler. • `92 - BIS-Fail-ServerBadParameter`: An internal error occurred while the service program was communicating with the BIS Request Handler. • `93 - BIS-Fail-ServerWrongMsg`: An internal error occurred while the service program was communicating with the BIS Request Handler. • `99 - BIS-Fail-ServerConnectionLost`: The connection between the BIS service program and the BIS Request Handler failed.

Service Program Functions

The following COBOL-callable functions may be used in BIS service programs to communicate with BIS. They are detailed in the following sections.

B_ReadRequest

This function call retrieves the current BIS request for processing by the service program. The syntax of this function call is:

```
Call "B_ReadRequest"
    [using TimeoutInSeconds[, RequestDocumentPointer, RequestDocumentLength]]
    giving BIS-Status.
```

When this function is called, execution of the service program is suspended until one of the following events occurs:

Event	Action
The BIS Request Handler renders an `XMLExchange` tag for the current session	This tag causes the current request data to be encoded into XML and placed into the file specified by the *BIS_FILENAME* environment variable, unless the `RequestDocumentPointer` parameter is present.

Event	Action
The BIS Request Handler renders a `StopService` or `SessionComplete` tag for the current session	This indicates that the service is no longer required. The service program should terminate and is granted [*ServiceTimeout*] seconds to do so.
The optional `TimeoutInSeconds` parameter expires	This timeout allows the BIS service program to regain control and perform some work. When complete, the program should call `B_ReadRequest` again.
The *`InactivityTimeout`* period expires	This indicates that the end user has abandoned the session. The service program should terminate and is granted 30 seconds to do so.

The most common result codes are as follows (see *BIS Return Codes* for a complete table):

BIS-Status Code	Event Description
`BIS-Success`	A valid request was received.
`BIS-Warn-RequestTimeExpired`	The `TimeoutInSeconds` parameter was specified and no request was received before the time elapsed.
`BIS-Warn-RequestOutstanding`	A request is outstanding. The service program must write a response before another request can be received. This code is only returned by BIS/Apache; BIS/IIS recreates the exchange file and returns `BIS_Success` each time that `B_ReadRequest` is called for the same request.
`BIS-Fail-SessionAbandoned`	Service termination is being requested because the BIS session inactivity time has elapsed without a request.
`BIS-Fail-SessionComplete`	Service termination is being requested because a `StopService` tag was rendered.
`BIS-Fail-ServiceComplete`	Service termination is being requested because a `SessionComplete` tag was rendered.

(These values are defined in file `bisdef.cpy`). Other codes may also be returned; see *BIS Return Codes*

When execution resumes and the result code is `BIS-Success`, the Request Document is returned one of two ways:

- If the `RequestDocumentPointer` parameter is omitted, the file specified by the *BIS_FILENAME* environment variable contains the request in XML format.
- If the `RequestDocumentPointer` parameter is present, the request is placed into memory and `RequestDocumentPointer` is set to point to that memory, replacing whatever value the pointer data item had before the call. Also, the length of the document is placed into `RequestDocumentLength`.

Notes

- The BIS Service Engine starts the service timer when this function returns. The program is then given *`ServiceTimeout`* seconds to process the request and perform one of these actions:
  - Call `B_WriteResponse`
  - Call `B_SetServiceTimeout`. In particular, a value of `0` resets the timer and starts another *ServiceTimeout* interval.
  - Terminate the program.

If the service program processes for more than the *ServiceTimeout* interval without performing one of the above functions, the BIS Service Engine assumes the service program is lost and begins termination processing (as if a `StopService` tag had been rendered).

- If specifying the optional `TimeoutInSeconds` parameter and a request does not arrive within the specified amount of time, the function returns a `BIS-Warn-RequestTimeExpired` status code. The program can then perform some processing and either exit or reissue the `B_ReadRequest`.

  Specifying a timeout value

  of `0` causes this function to return immediately unless a request is waiting. The routine use of a timeout value of `0` to poll for requests is strongly discouraged as it may significantly impact server performance.
- If `TimeoutInSeconds` is not specified, this function does not return until one of the other termination events occur (that is, the default timeout is infinite).
- If `RequestDocumentPointer` is specified, `RequestDocumentLength` must also be specified.
- If the request document pointer and length are returned in `RequestDocumentPointer` and `RequestDocumentLength`, this pointer and length are normally used in an XML IMPORT TEXT statement to obtain the request information into a COBOL data structure for further processing.
- The memory area referred to by `RequestDocumentPointer` after the call to `B_ReadRequest` belongs to BIS and the COBOL program must not attempt to free that memory.
- When specifying the `RequestDocumentPointer` for the default timeout value, the TimeoutInSeconds argument may be specified as OMITTED, since the `RequestDocumentPointer` argument must be the second USING argument to `B_ReadRequest`.

B_WriteResponse

This function call transmits a response document to the BIS Request Handler to be inserted into the output stream, replacing the `XMLExchange` tag in the output stream. The response document must be written into the request file (specified by the *BIS_FILENAME* environment variable) or placed into a memory area before `B_WriteResponse` is called unless session or service termination is being requested-in this case, the response is optional (see below).

The response file typically contains an requested HTML or XML that is inserted into the `.srf` file. It may also contain a SOAP result or anything else that is meaningful to the client program that issued the request.

The syntax of this function call is:

```
Call "B_WriteResponse"
   [using ProgramDisposition
   [, ResponseDocumentPointer, ResponseDocumentLength]]
    giving BIS-Status.
```

If this is the final call to `B_WriteResponse` by this service, the optional `ProgramDisposition` parameter may be used to indicate that the service program is finished, if the session should be destroyed, and if there is a payload that should be rendered into the response. Here are the values:

```
 78 BIS-Response-Normal                  Value 0. *> Default normal response
 78 BIS-Response-ServiceComplete         Value 1. *> End program only
 78 BIS-Response-SessionComplete         Value 2. *> End program and session
*78*BIS-Response-RecycleService          Value 3. *> RESERVED FOR FUTURE USE
```

The ProgramDisposition codes descriptions are as follows:

BIS-Status Code	Event Description
`BIS-Response-Normal`	The default is that BIS makes no assumptions about what the service program will do next. However, the service program is granted only 30 seconds to exit or to read the next request.

BIS-Status Code	Event Description
BIS-Response-ServiceComplete	Indicates that the service has completed processing, cannot accept any additional requests, and is about to terminate. The XML output is written to the output stream, completely replacing the XMLExchange tag. The service program is then disconnected from the session and allowed to run to completion in the background. If the service program subsequently calls B_ReadRequest, it receives a BIS-Fail-ServiceComplete error status. Note that the session is not destroyed, and if a StartService tag is executed before the session expires, the new service program runs under the current session. This is logically the same as processing a StopService tag in the .srf file.
BIS-Response-SessionComplete	Indicates that the both the service and the session are complete and cannot not accept any additional requests. The XML output is written to the output stream, completely replacing the XMLExchange tag. The service program is disconnected from the session and allowed to run to completion in the background. If the service program subsequently calls B_ReadRequest, it receives a BIS-Fail-SessionComplete error status. Note that, in this case, the session is destroyed, and a new session is created if another .srf file is requested. This is logically the same as processing a SessionComplete tag in the .srf file.
BIS-Response-RecycleService	Reserved for future use.

The Response Document is delivered in one of two ways:

- In the file specified by the BIS_FILENAME environment variable, in which case the ResponseDocumentPointer and ResponseDocumentLength parameters must be omitted.
- In a memory area, in which case the ResponseDocumentPointer must point to the memory area and ResponseDocumentLength must contain the length of the document, and these two parameters must be present in the call to B_WriteResponse.

The BIS-Status result field and the result codes are defined in bisdef.cpy. Here are the most common return codes:

BIS-Status Code	Event Description
BIS-Success	The BIS Request Handler accepted the response. This does not mean that it was delivered to the user agent. However, the service program should presume success and resume processing.
BIS-Warn-ResponseUnexpected	There is no pending request to respond to.

Notes

- Unlike B_ReadRequest, this call returns as soon the BIS Request Handler accepts the document. This function call does not block waiting for a response from BIS.
- After writing a response, the service program will normally either call B_ReadRequest or terminate.

- The BIS Service Engine starts the service timer when this function returns. The program has 30 seconds to perform one of these actions:
  - Call `B_ReadRequest`.
  - Call `B_SetServiceTimeout`. A parameter of `0` restarts the service timer.
  - Terminate.

  If the service program processes for more than 30 seconds without performing one of the above functions, the BIS Service Engine assumes the service program is lost and begins termination processing (as if a `StopService` tag had been rendered).
- Other codes may also be returned, but that normally indicates a serious problem has occurred.
- By default, the response is sent back to the client with the HTTP status of `OK`, which is the value 200. However, the function `B_SetResponseStatus` may be used to alter the HTTP status returned.
- If `ResponseDocumentPointer` is present, then `ResponseDocumentLength` must also be specified.
- `ResponseDocumentPointer`'s pointer value, and `ResponseDocumentLength` document length, are normally obtained by using an XML EXPORT TEXT statement to create the response document from a COBOL data structure in the COBOL service program; in that case, it is the COBOL program's responsibility to free the response memory area, after calling `B_WriteResponse`, by using an XML FREE TEXT statement that specifies the `ResponseDocumentPointer` argument.
- When specifying the `ResponseDocumentPointer` for the default program disposition, the `ProgramDisposition` argument may be specified as OMITTED since the `ResponseDocumentPointer` argument must be the second USING argument to `B_WriteResponse`.

B_Exchange

This function call is equivalent to calling `B_WriteResponse` immediately followed by `B_ReadRequest`. This function is deprecated and should not be used because it is not possible to specify the program disposition parameter.

Note: The `B_Exchange` function call is not supported by BIS for Visual COBOL and is deprecated for BIS for AcuCOBOL-GT.

The syntax of this function call is:

```
Call "B_Exchange" using TimeoutInSeconds giving BIS-Status.
```

This is logically equivalent to this sequence:

```
call "B_WriteResponse" giving BIS-Status
if BIS-Status = BIS-Success or BIS-Status = BIS-Warn-ResponseUnexpected then
    call "B_ReadRequest" using TimeoutInSeconds giving BIS-Status
endif
```

If only `B_Exchange` calls are used in a service program, the first call to `B_WriteResponse` is performed in the absence of a request because `B_ReadRequest` has not yet been called. An error will be returned. This error is ignored and the result code of the call to `B_Exchange` reflects the result of the `B_ReadRequest`.

See the description of *B_ReadRequest* for a table of result codes and their interpretation.

B_SetInactivityTimeout

This function allows the service program to control the length of time that BIS waits for a request before considering a session to be abandoned.

A timer is started in a session when each request is processed for that session. If a new request is not received before the timer elapses, any active services in that session are terminated and the session is terminated.

If a request is subsequently received for a terminated session, BIS creates a new session.

The syntax of this function call is:

```
Call "B_SetInactivityTimeout" using TimeoutInSeconds giving BIS-Status.
```

where *TimeoutInSeconds* may be:

- The actual number of seconds this session waits for a new request. Valid values range from 10 to 3600 seconds. All values out of this range other than `0` are treated as if `-1` was specified.
- `0` to restart the inactivity timer without changing the number of seconds allowed between requests.
- `-1` to reset the timeout value (on BIS/IIS) to the default value of 600 seconds or 10 minutes, or the default inactivity timeout configured in `/etc/xbis.conf` (on BIS/Apache).

The **BIS-Status** result field and the result codes are defined in `BISDEF.CPY`. Here are the most common return codes:

	Event Description
`BIS-Success`	The call is successful.
`BIS-Fail-SessionAbandoned`	Service termination is already being requested because the BIS session inactivity timeout period has elapsed without a request. This function call has no effect.
`BIS-Fail-SessionComplete`	Service termination is requested because a `SessionComplete` tag was rendered. This function call had no effect.
`BIS-Fail-ServiceComplete`	Service termination is requested because a `StopService` tag was rendered. This function call had no effect.

Notes

- The default inactivity timeout period is 600 seconds (10 minutes). *Session Inactivity Timeout* describes how to change the default for all BIS sessions on this server.
- The inactivity timeout can also be set in a `.srf` file with the `SessionParms` tag.
- All calls to this function restart the timer. Specify `0` to restart the timer without changing the value currently in effect.
- BIS/IIS defers processing this function until an `XMLExchange` tag is processed. The main implication of this restriction is that if the client starts the program and then browses pages that do not include an `XMLExchange` tag while the program calls `B_SetInactivityTimeout()` followed by `B_ReadRequest()`, the updated inactivity timeout interval does not take effect until an `XMLExchange` tag is processed. This is an unlikely scenario because there is no reason to start a service program if an `XMLExchange` tag is not imminent.

B_SetServiceTimeout

This function allows the service program to control the length of time that the service program is permitted to run without interacting with BIS.

The service timer is reset when:

- The service program is started.
- The service program calls any B_ function.

If the timer elapses, the service program is terminated. The default service timeout interval is 30 seconds.

The syntax of this function call is:

```
Call "B_SetServiceTimeout" using TimeoutInSeconds giving BIS-Status.
```

where `TimeoutInSeconds` may be:

- The actual number of seconds allowed between calls to BIS B_ functions. Note that the value may range from 10 to 3600 seconds (1 hour). All values out of this range other than 0 are treated as if -1 was specified.
- 0 to restart the service timer without changing the number of seconds allowed between calls to B_ functions.
- -1 to reset the timeout value (on BIS/IIS) to the default value of 30 seconds or (on BIS/Apache) the default service timeout configured in /etc/xbis.conf.

The BIS-Status result field and the result codes are defined in BISDEF.CPY. Here are the most common return codes:

BIS-Status Code	Event Description
BIS-Success	The call was successful.
BIS-Fail-SessionAbandoned	Service termination is already being requested because the BIS session inactivity time has elapsed without a request. This function call had no effect.
BIS-Fail-SessionComplete	Service termination is already being requested because a SessionComplete tag was rendered. This function call had no effect.
BIS-Fail-ServiceComplete	Service termination is already being requested because a StopService tag was rendered. This function call had no effect.

Notes

- The default service timeout period is 30 seconds. The section titled *Service Timeouts* describes how the default may be changed for all BIS services on this server.
- The service timeout may also be set in a .srf file with the SessionParms tag.
- All calls to this function will restart the timer. Specify 0 to restart the timer without changing the value currently in effect.
- BIS/IIS defers processing of this function until an XMLExchange tag is processed. The main implication of this restriction is that if the client starts the program and then browses pages that do not include an XMLExchange tag while the program calls B_SetServiceTimeout() then B_ReadRequest(), the updated service timeout interval does not take effect until an XMLExchange tag is processed. This is an unlikely but possible scenario because there is no reason to start a service program if an XMLExchange tag is not imminent.

B_SetResponseStatus

This function allows the service program to control the HTTP status that is returned with a response. By default, the response status will be 200, which is an HTTP status of OK. However, if it is desirable to return a different status, this function may be used to alter the status for the next response. Subsequent responses return OK again unless B_SetResponseStatus is called before B_WriteResponse.

The syntax of this function call is:

```
Call "B_SetResponseStatus" using ResponseStatus giving BIS-Status.
```

where *ResponseStatus* may be:

- An HTTP status code detailed at *http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html#sec6.1*. In general, one of the 200, 300 or 400 codes should be used to indicate the result of the operation. Avoid the 500 codes . They are for errors detected by the web server.
- The following table describes the HTTP status codes specified in RFC 2616, their status names, and a synopsis of the RFC 2616 description.

Status Code	Status Name	HTTP Standard Description
1*xx*	Informational	This class of status code indicates a provisional response, consisting only of the Status-Line and optional headers, and is terminated by an empty line. There are no required headers for this class of status code. A client must be prepared to accept one or more 1*xx* status responses prior to a regular response, even if the client does not expect a 100 (Continue) status message. Unexpected 1*xx* status responses may be ignored by a user agent.
100	Informational	The client should continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server. The client should continue by sending the remainder of the request or, if the request has already been completed, ignore this response. The server must send a final response after the request has been completed. This status code is generated by the web server directly and should not be returned by the service program.
101	Switching Protocols	The server understands and is willing to comply with the client's request, via the Upgrade message header field, for a change in the application protocol being used on this connection. The server will switch protocols to those defined by the response's Upgrade header field immediately after the empty line which terminates the 101 response. This status code is generated by the web server directly and should not be returned by the service program.
2*xx*	Successful	This class of status code indicates that the client's request was successfully received, understood, and accepted.
200	OK	The request has succeeded. The information returned with the response is dependent on the method used in the request.
201	Created	The request has been fulfilled and resulted in a new resource being created. The newly created resource can be referenced by the URI(s) returned in the entity of the response, with the most specific URI for the resource given by a Location header

Status Code	Status Name	HTTP Standard Description
		field. The response should include an entity containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. The origin server must create the resource before returning the 201 status code. If the action cannot be carried out immediately, the server should respond with 202 (Accepted) response instead.
202	Accepted	The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place. There is no facility for resending a status code from an asynchronous operation such as this. The 202 response is intentionally non-committal. Its purpose is to allow a server to accept a request for some other process (perhaps a batch-oriented process that is only run once per day) without requiring that the user agent's connection to the server persist until the process is completed. The entity returned with this response should include an indication of the request's current status and either a pointer to a status monitor or some estimate of when the user can expect the request to be fulfilled.
203	Non-Authoritative Information	The returned meta information in the entity-header is not the definitive set as available from the origin server, but is gathered from a local or a third-party copy. The set presented may be a subset or superset of the original version. For example, including local annotation information about the resource might result in a superset of the meta information known by the origin server. Use of this response code is not required and is only appropriate when the response would otherwise be 200 (OK).
204	No Content	The server has fulfilled the request but does not need to return an entity-body, and might want to return updated meta information. The response may include new or updated meta information in the form of entity-headers, which if present

Status Code	Status Name	HTTP Standard Description
		should be associated with the requested variant. If the client is a user agent, it should not change its document view from that which caused the request to be sent. This response is primarily intended to allow input for actions to take place without causing a change to the user agent's active document view, although any new or updated metadata should be applied to the document currently in the user agent's active view. The 204 response must not include a message-body, and thus is always terminated by the first empty line after the header fields.
205	`Reset Content`	The server has fulfilled the request and the user agent should reset the document view which caused the request to be sent. This response is primarily intended to allow input for actions to take place via user input, followed by a clearing of the form in which the input is given so that the user can easily initiate another input action. The response must not include an entity.
206	`Partial Content`	The server has fulfilled the partial GET request for the resource.
3*xx*	`Redirection`	This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request. The action required may be carried out by the user agent without interaction with the user if and only if the method used in the second request is GET or HEAD. A client should detect infinite redirection loops, since such loops generate network traffic for each redirection.
300	`Multiple Choices`	The requested resource corresponds to any one of a set of representations, each with its own specific location, and agent- driven negotiation information is being provided so that the user (or user agent) can select a preferred representation and redirect its request to that location. Unless it was a HEAD request, the response should include an entity containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the

Status Code	Status Name	HTTP Standard Description
		media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice may be performed automatically. However, this specification does not define any standard for such automatic selection. If the server has a preferred choice of representation, it should include the specific URI for that representation in the Location field; user agents may use the Location field value for automatic redirection. This response can be cached unless otherwise indicated.
301	`Moved Permanently`	The requested resource has been assigned a new permanent URI and any future references to this resource should use one of the returned URIs. Clients with link editing capabilities ought to automatically re-link references to the Request-URI to one or more of the new references returned by the server, where possible. This response can be cached unless otherwise indicated. The new permanent URI should be given by the Location field in the response. Unless the request method was HEAD, the entity of the response should contain a short hypertext note with a hyperlink to the new URI(s). If the 301 status code is received in response to a request other than GET or HEAD, the user agent must not automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.
302	`Found`	The requested resource resides temporarily under a different URI. Since the redirection might be altered on occasion, the client should continue to use the Request-URI for future requests. This response can only be cached if indicated by a Cache-Control or Expires header field. The temporary URI should be given by the Location field in the response. Unless the request method was HEAD, the entity of the response should contain a short hypertext note with a hyperlink to the new URI(s).

Status Code	Status Name	HTTP Standard Description
		If the 302 status code is received in response to a request other than GET or HEAD, the user agent must not automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.
303	See Other	The response to the request can be found under a different URI and should be retrieved using a GET method on that resource. This method exists primarily to allow the output of a POST-activated script to redirect the user agent to a selected resource. The new URI is not a substitute reference for the originally requested resource. The 303 response must not be cached, but the response to the second (redirected) request may be cached. The different URI should be given by the Location field in the response. Unless the request method was HEAD, the entity of the response should contain a short hypertext note with a hyperlink to the new URI(s).
304	Not Modified	If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server should respond with this status code. The 304 response must not contain a message-body, and thus is always terminated by the first empty line after the header fields.
305	Use Proxy	The requested resource must be accessed through the proxy given by the Location field. The Location field gives the URI of the proxy. The recipient is expected to repeat this single request via the proxy. 305 responses must only be generated by origin servers.
306	(unused)	The 306 status code was used in a previous version of the specification, is no longer used, and the code is reserved.
307	Temporary Redirect	The requested resource resides temporarily under a different URI. Since the redirection may be altered on occasion, the client should continue to use the Request-URI for future requests. This response can only be cached if indicated by a

Status Code	Status Name	HTTP Standard Description
		Cache-Control or Expires header field. The temporary URI should be given by the Location field in the response. Unless the request method was HEAD, the entity of the response should contain a short hypertext note with a hyperlink to the new URI(s) , since many pre-HTTP/1.1 user agents do not understand the 307 status. Therefore, the note should contain the information necessary for a user to repeat the original request on the new URI. If the 307 status code is received in response to a request other than GET or HEAD, the user agent must not automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.
4*xx*	`Client Error`	The 4*xx*class of status code is intended for cases in which the client seems to have erred. Except when responding to a HEAD request, the server should include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. These status codes are applicable to any request method. User agents should display any included entity to the user. If the client is sending data, a server implementation using TCP should be careful to ensure that the client acknowledges receipt of the packet(s) containing the response, before the server closes the input connection. If the client continues sending data to the server after the close, the server's TCP stack will send a reset packet to the client, which may erase the client's unacknowledged input buffers before they can be read and interpreted by the HTTP application.
400	`Bad Request`	The request could not be understood by the server due to malformed syntax. The client should not repeat the request without modifications.
401	`Unauthorized`	The request requires user authentication. The response must include a WWW-Authenticate header field containing a challenge applicable to the requested resource. The client may repeat the request

Status Code	Status Name	HTTP Standard Description
		with a suitable Authorization header field. If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials. If the 401 response contains the same challenge as the prior response, and the user agent has already attempted authentication at least once, then the user should be presented the entity that was given in the response, since that entity might include relevant diagnostic information.
402	`Payment Required`	This code is reserved for future use.
403	`Forbidden`	The server understood the request, but is refusing to fulfill it. Authorization will not help and the request should not be repeated. If the request method was not HEAD and the server wishes to make public why the request has not been fulfilled, it should describe the reason for the refusal in the entity. If the server does not wish to make this information available to the client, the status code 404 (`Not Found`) can be used instead.
404	`Not Found`	The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent. The 410 (`Gone`) status code should be used if the server knows, through some internally configurable mechanism, that an old resource is permanently unavailable and has no forwarding address. This status code is commonly used when the server does not wish to reveal exactly why the request has been refused, or when no other response is applicable.
405	`Method Not Allowed`	The method specified in the Request-Line is not allowed for the resource identified by the Request-URI. The response must include an Allow header containing a list of valid methods for the requested resource.
406	`Not Acceptable`	The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request. Unless it was a HEAD request, the response should include an entity

Status Code	Status Name	HTTP Standard Description
		containing a list of available entity characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice may be performed automatically. However, this specification does not define any standard for such automatic selection. If the response could be unacceptable, a user agent should temporarily stop receipt of more data and query the user for a decision on further actions.
407	`Proxy Authentication Required`	This code is similar to 401 (`Unauthorized`), but indicates that the client must first authenticate itself with the proxy. The proxy must return a Proxy-Authenticate header field containing a challenge applicable to the proxy for the requested resource. The client may repeat the request with a suitable Proxy-Authorization header field.
408	`Request Timeout`	The client did not produce a request within the time that the server was prepared to wait. The client may repeat the request without modifications at any later time.
409	`Conflict`	The request could not be completed due to a conflict with the current state of the resource. This code is only allowed in situations where it is expected that the user might be able to resolve the conflict and resubmit the request. The response body should include enough information for the user to recognize the source of the conflict. Ideally, the response entity would include enough information for the user or user agent to fix the problem; however, that might not be possible and is not required. Conflicts are most likely to occur in response to a PUT request. For example, if versioning were being used and the entity being PUT included changes to a resource which conflict with those made by an earlier (third-party) request, the server might use the 409 response to indicate that it can't complete the request. In this case, the response entity would likely

Status Code	Status Name	HTTP Standard Description
		contain a list of the differences between the two versions in a format defined by the response Content-Type.
410	Gone	The requested resource is no longer available at the server and no forwarding address is known. This condition is expected to be considered permanent. Clients with link editing capabilities should delete references to the Request-URI after user approval. If the server does not know, or has no facility to determine, whether or not the condition is permanent, the status code 404 (Not Found) should be used instead. This response can be cached unless indicated otherwise. The 410 response is primarily intended to assist the task of web maintenance by notifying the recipient that the resource is intentionally unavailable and that the server owners desire that remote links to that resource be removed. Such an event is common for limited-time, promotional services and for resources belonging to individuals no longer working at the server's site. It is not necessary to mark all permanently unavailable resources as *gone* or to keep the mark for any length of time -- that is left to the discretion of the server owner.
411	Length Required	The server refuses to accept the request without a defined Content-Length. The client may repeat the request if it adds a valid Content-Length header field containing the length of the message-body in the request message.
412	Precondition Failed	The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server. This response code allows the client to place preconditions on the current resource meta information (header field data) and thus prevent the requested method from being applied to a resource other than the one intended.
413	Request Entity Too Large	The server is refusing to process a request because the request entity is larger than the server is willing or able to process. The server may close the connection to prevent the client from continuing the request.

Status Code	Status Name	HTTP Standard Description
		If the condition is temporary, the server should include a Retry- After header field to indicate that it is temporary and after what time the client may try again.
414	`Request-URI Too Long`	The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret. This rare condition is only likely to occur when a client has improperly converted a POST request to a GET request with long query information, when the client has descended into a URI *black hole* of redirection (e.g., a redirected URI prefix that points to a suffix of itself), or when the server is under attack by a client attempting to exploit security holes present in some servers using fixed-length buffers for reading or manipulating the Request-URI.
415	`Unsupported Media Type`	The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.
416	`Requested Range Not Satisfiable`	A server should return a response with this status code if a request included a Range request-header field, and none of the range-specifier values in this field overlap the current extent of the selected resource, and the request did not include an If-Range request-header field. (For byte-ranges, this means that the first-byte-pos of all of the byte-range-spec values were greater than the current length of the selected resource.) When this status code is returned for a byte-range request, the response should include a Content-Range entity-header field specifying the current length of the selected resource. This response must not use the multi-part/byteranges content-types.
417	`Expectation Failed`	The expectation given in an Expect request-header field could not be met by this server, or, if the server is a proxy, the server has unambiguous evidence that the request could not be met by the next-hop server.
5*xx*	`Server Error`	Response status codes beginning with the digit 5 indicate cases in which the server is aware that it has erred or is incapable of performing

Status Code	Status Name	HTTP Standard Description
		the request. Except when responding to a HEAD request, the server should include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents should display any included entity to the user. These response codes are applicable to any request method. In general, these error codes are generated by HTTP server itself, and the service program should not use them.
500	`Internal Server Error`	The server encountered an unexpected condition which prevented it from fulfilling the request.
501	`Not Implemented`	The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any resource.
502	`Bad Gateway`	The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.
503	`Service Unavailable`	The server is currently unable to handle the request due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated after some delay. If known, the length of the delay may be indicated in a Retry-After header. If no Retry-After is given, the client should handle the response as it would for a 500 response.
504	`Gateway Timeout`	The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server specified by the URI (e.g. HTTP, FTP, LDAP) or some other auxiliary server (e.g. DNS) it needed to access in attempting to complete the request.
505	`HTTP Version Not Supported`	The server does not support, or refuses to support, the HTTP protocol version that was used in the request message. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client, other than with this error message. The response should contain an entity describing why that version is not

Status Code	Status Name	HTTP Standard Description
		supported and what other protocols are supported by that server.

The **BIS-Status** result field and the result codes are defined in `BISDEF.CPY`. Here are the most common return codes:

BIS-Status Code	Event Description
`BIS-Success`	The call was successful.

Notes

- In general, a SOAP-base web service should imbed its response status inside the SOAP response and let the Request Handler manage the HTTP Status.
- This function call is of most use to a REST-based web service where using the native URL and status codes of HTTP are encouraged.
- Many of the status codes are intended to control the interaction of the web server and a browser and the code may be used safely to communicate between the service program and a web client. Other codes are intended for consumption by proxy servers (or are intended to be generated by them) and should be avoided.
- The table of status codes is given here as a reference and is not intended to be a substitute for RFC 2616.

B_Trace

This function allows the service program to write messages into the BIS trace file.

The syntax of this function call is:

```
Call "B_Trace" using Item[, Item]… giving BIS-Status.
```

where *`Item`* may be any data item that can be converted to a displayable value and written into the trace buffer.

The **BIS-Status** result field and the result codes are defined in `BISDEF.CPY`. Here are the most common return codes:

BIS-Status Code	Event Description
`BIS-Success`	The call was successful.

Notes

- If an item's displayable value fits into `B_Trace`'s remaining trace buffer, the displayable value is simply copied into the buffer.
- If an item's displayable value does not fit into `B_Trace`'s remaining trace buffer, but is shorter than the trace buffer when empty, the current trace buffer is written into the trace and the item's displayable value is written into the empty buffer.
- If an item's displayable value does not fit into `B_Trace`'s remaining trace buffer, and does not fit into the trace buffer when empty, the displayable value is copied into the trace buffer until it fills, and then the trace buffer is written to the trace. The item's remaining displayable value is then written to the trace, in trace buffer length portions, until the remainder can be written into the trace buffer without filling it.
- After all Items have been processed, the trace buffer, if not empty, is written into the trace.

Server Variables Reference

The following table describes the server variables that may be inspected with the `Value` tag. Note that the descriptions are taken from Microsoft's IIS SDK documentation and not all server variables are displayed in the `TRACE` output if empty.

Variable	Platform	Description
`ALL_HTTP`	IIS	All `HTTP` headers sent by the client.
`ALL_RAW`	IIS	Retrieves all headers in raw form. The difference between `ALL_RAW` and `ALL_HTTP` is that `ALL_HTTP` places an `HTTP_` *`prefix`* before the header name and the header name is always capitalized. In `ALL_RAW` the header name and values appear as they are sent by the client.
`APP_POOL_ID`	IIS	Returns the name of the application pool that is running in the IIS worker process that is handling the request.
`APPL_MD_PATH`	IIS	Retrieves the metabase path for the Application for the BIS server
`APPL_PHYSICAL_PATH`	IIS	Retrieves the physical path corresponding to the metabase path. IIS converts the `APPL_MD_PATH` to the physical (directory) path to return this value.
`AUTH_PASSWORD`	IIS	The value entered in the client's authentication dialog box. This variable is available only if Basic authentication is used.
`AUTH_TYPE`	IIS	The authentication method that the server uses to validate users when they attempt to access a protected script.
`AUTH_USER`	IIS	The name of the user as it is derived from the authorization header sent by the client, before the user name is mapped to a Windows account. This variable is no different from `REMOTE_USER`. If you have an authentication filter installed on your web server that maps incoming users to accounts, use `LOGON_USER` to view the mapped user name.
`CERT_COOKIE`	IIS	Unique ID for the client certificate, returned as a string. This can be used as a signature for the whole client certificate.
`CERT_FLAGS`	IIS	Bit `0` set to `1` if the client certificate is present.

Variable	Platform	Description
		Bit 1 is set to 1 if the certification authority of the client certificate is invalid (that is, it is not in the list of recognized certification authorities on the server).
CERT_ISSUER	IIS	Issuer field of the client certificate (O=MS, OU=IAS, CN=user name, C=USA).
CERT_KEYSIZE	IIS	Number of bits in the Secure Sockets Layer (SSL) connection key size. For example, 128.
CERT_SECRETKEYSIZE	IIS	Number of bits in server certificate private key. For example, 1024.
CERT_SERIALNUMBER	IIS	Serial number field of the client certificate.
CERT_SERVER_ISSUER	IIS	Issuer field of the server certificate.
CERT_SERVER_SUBJECT	IIS	Subject field of the server certificate.
CERT_SUBJECT	IIS	Subject field of the client certificate.
CONTENT_LENGTH	All	The length of the content as given by the client.
CONTENT_TYPE	All	The data type of the content. Used with queries that have attached information, such as the HTTP queries GET, POST, and PUT.
DOCUMENT_ROOT	Apache	Contains the local directory from which the server is serving pages.
GATEWAY_INTERFACE	All	The revision of the CGI specification used by the server. The format is CGI/*revision*. Example: CGI/1.1.
HTTP_*HeaderName*	All	The value stored in the HTTP header *HeaderName* . Any header other than those listed below must be preceded by HTTP in order for the Value(*variable*, Server) collection to retrieve its value. This is useful for retrieving custom headers. The server interprets any underscore (_) characters in *HeaderName* as dashes in the actual header. For example, if you specify HTTP_MY_HEADER, the server searches for a request header named MY-HEADER .
HTTP_ACCEPT	All	Returns the value of the Accept header. For example, image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel.

Variable	Platform	Description
HTTP_ACCEPT_CHARSET	IIS	The raw contents of the Accept-Charset header: contains a list of character sets that are acceptable in the response. For example, iso-8859-5, unicode-1-1;q=0.8
HTTP_ACCEPT_ENCODING	All	The raw contents of the Accept-Encoding header: contains a list of accepted encoding types, for example, gzip, deflate.
HTTP_ACCEPT_LANGUAGE	All	The raw contents of the Accept-Language header
HTTP_AUTHORIZATION	IIS	The raw contents of the Authorization header.
HTTP_CACHE_CONTROL	All	The raw contents of the Cache-Control header.
HTTP_CONNECTION	All	The raw contents of the Connection header.
HTTP_CONTENT_LENGTH	IIS	The raw contents of the Content-Length header.
HTTP_CONTENT_TYPE	IIS	The raw contents of the Content-Type header.
HTTP_COOKIE	All	Returns the cookie string that was included with the request.
HTTP_DATE	IIS	The raw contents of the Date header.
HTTP_EXPECT	IIS	The raw contents of the Expect header.
HTTP_FROM	IIS	The raw contents of the From header.
HTTP_HOST	All	Returns the name of the web server. This may or may not be the same as SERVER_NAME, depending on type of name resolution you are using on your web server (IP address or host header).
HTTP_IF_MATCH	All	The raw contents of the If-Match header.
HTTP_IF_MODIFIED_SINCE	IIS	The raw contents of the If-Modified-Since header.
HTTP_IF_NONE_MATCH	IIS	The raw contents of the If-None-Match header.
HTTP_IF_RANGE	IIS	The raw contents of the If-Range header.
HTTP_IF_UNMODIFIED_SINCE	IIS	The raw contents of the If-Unmodified-Since header.

Variable	Platform	Description
HTTP_MAX_FORWARDS	IIS	The raw contents of the Max-Forwards header.
HTTP_PRAGMA	IIS	The raw contents of the Pragma header.
HTTP_PROXY_AUTHORIZATION	IIS	The raw contents of the Proxy-Authorization header.
HTTP_RANGE	IIS	The raw contents of the Range header.
HTTP_REFERER	All	Returns a string that contains the URL of the page that referred the request to the current page by using an HTML <A> tag. Note that the URL is the one that the user typed into the browser address bar, which may not include the name of a default document. If the page is redirected, HTTP_REFERER is empty. HTTP_REFERER is not a mandatory member of the HTTP specification and some clients allow the end user to disable this information. Note that, in this case, REFERER is spelled with a single R.
HTTP_TE	All	The raw contents of the TE header.
HTTP_TRAILER	All	The raw contents of the Trailer header.
HTTP_TRANSFER_ENCODING	All	The raw contents of the Transfer-Encoding header.
HTTP_UPGRADE	All	The raw contents of the Upgrade header.
HTTP_URL	All	Returns the raw, encoded URL. Example: /xbis/default.srf?query. Note that the scheme and host name are not part of this URL. On Apache, this does not include the query portion.
HTTP_USER_AGENT	All	Returns a string describing the browser that sent the request.
HTTP_VERSION	IIS	The raw contents of the Version header.
HTTP_VIA	IIS	The raw contents of the Via header.
HTTP_WARNING	IIS	The raw contents of the Warning header.
HTTPS	All	Returns ON if the request came in through a secure channel (for example, SSL); or it returns OFF, if the request is for an insecure channel.

Variable	Platform	Description
HTTPS_KEYSIZE	IIS	Number of bits in the SSL connection key size. For example, 128.
HTTPS_SECRETKEYSIZE	IIS	Number of bits in the server certificate private key. For example, 1024.
HTTPS_SERVER_ISSUER	IIS	Issuer field of the server certificate.
HTTPS_SERVER_SUBJECT	IIS	Subject field of the server certificate.
INSTANCE_ID	IIS	The ID for the IIS instance in textual format. If the instance ID is 1, it appears as a string. You can use this variable to retrieve the ID of the web server instance (in the metabase) to which the request belongs.
INSTANCE_META_PATH	IIS	The metabase path for the instance of IIS that responds to the request.
LOCAL_ADDR	IIS	Returns the server address on which the request came in. This is important on computers where there can be multiple IP addresses bound to the computer, and you want to find out which address the request used.
LOGON_USER	IIS	The Windows account that the user is impersonating while connected to your web server. Use REMOTE_USER, UNMAPPED_REMOTE_USER, or AUTH_USER to view the raw user name that is contained in the request header. The only time LOGON_USER holds a different value than these other variables is if you have an authentication filter installed.
PATH	Apache	Contains the Apache Server's PATH environment variable.
PATH_INFO	IIS	Extra path information, as given by the client. You can access scripts by using their virtual path and the PATH_INFO server variable. If this information comes from a URL, it is decoded by the server before it is passed to the CGI script.
PATH_TRANSLATED	IIS	A translated version of PATH_INFO that takes the path and performs any necessary virtual-to-physical mapping.
QUERY_STRING	All	Query information stored in the string following the question mark (?) in the HTTP request.
REMOTE_ADDR	All	The IP address of the remote host that is making the request.
REMOTE_HOST	IIS	The name of the host that is making the request. If the server does not

Variable	Platform	Description
		have this information, it will set REMOTE_ADDR and leave this empty.
REMOTE_PORT	All	The client port number of the TCP connection.
REMOTE_USER	IIS	The name of the user as it is derived from the authorization header sent by the client, before the user name is mapped to a Windows account. If you have an authentication filter installed on your web server that maps incoming users to accounts, use LOGON_USER to retrieve the mapped user name.
REQUEST_METHOD	All	The method used to make the request. For HTTP, this can be GET, HEAD, POST, and so on.
REQUEST_URI	Apache	The complete URI of the request.
SCRIPT_FILENAME	Apache	The complete file name of the script being executed.
SCRIPT_NAME	All	A virtual path to the script being executed. This is used for self-referencing URLs.
SERVER_ADDR	Apache	The IP address to which the request was sent.
SERVER_ADMIN	Apache	Contains the email address of the server's system administrator. (This is contents of the ServerAdmin configuration record.)
SERVER_NAME	All	The server's host name, DNS alias, or IP address as it would appear in self-referencing URLs.
SERVER_PORT	All	The server port number to which the request was sent.
SERVER_PORT_SECURE	IIS	A string that contains either 0 or 1. If the request is being handled on the secure port, then this is 1. Otherwise, it is 0.
SERVER_PROTOCOL	All	The name and revision of the request information protocol. The format is *protocol/revision* . Example: HTTP/1.1.
SERVER_SIGNATURE	Apache	The name and version of the Apache web server, plus the network name and port number on which the web server is running. Example: Apache/2.0.55 (Unix) mod_ssl/2.0.55 OpenSSL/ 0.9.8a Server at arokh.liant.com Port 80

Variable	Platform	Description
SERVER_SOFTWARE	All	The name and version of the server software that answers the request and runs the gateway. The format is *name/version*. Example: `Microsoft-IIS/5.0`
SSL_CIPHER	Apache HTTPS	The name of the SSL cipher in use. Example: `RC4-MD5`
SSL_CIPHER_EXPORT	Apache HTTPS	Contains `true` if the cipher is an export cipher and `false` otherwise.
SSL_CIPHER_ALGKEYSIZE	Apache HTTPS	The maximum number of bits permitted in the cipher's. Example: `128`
SSL_CIPHER_USEKEYSIZE	Apache HTTPS	The number of bits actually in use in the cipher. Example: `128`
SSL_CLIENT_A_KEY	Apache HTTPS	The signature algorithm used in the client key. Example: `rsaEncryption`
SSL_CLIENT_A_SIG	Apache HTTPS	The signature algorithm used in the client certificate. Example: `sha1WithRSAEncryption`
SSL_CLIENT_I_DN	Apache HTTPS	The client certificate issuer distinguish name subject. Example: `/CN=neo`
SSL_CLIENT_I_DN_CN	Apache HTTPS	The computer name of the client certificate issuer distinguish name subject. Example: `neo`
SSL_CLIENT_M_VERSION	Apache HTTPS	The client certificate's version. Example: `3`
SSL_CLIENT_M_SERIAL	Apache HTTPS	The client certificate's serial number. Example: `1DFD43180000000000015`
SSL_CLIENT_S_DN	Apache HTTPS	The client certificate distinguished name subject. Example: `/C=US/ST=TX/L=Austin/O=Liant/OU=R&D/CN=Mike Schultz/emailAddress=michael.schultz@microfocus.com`
SSL_CLIENT_S_DN_C	Apache HTTPS	The country of the client certificate distinguished name subject. Example: `US`
SSL_CLIENT_S_DN_CN	Apache HTTPS	The contact of the client certificate distinguished name subject. Example: `Mike Schultz`
SSL_CLIENT_S_DN_Email	Apache HTTPS	The email address of the client certificate distinguished name subject. Example: `michael.schultz@microfocus.com`

Variable	Platform	Description
SSL_CLIENT_S_DN_L	Apache HTTPS	The location of the client certificate distinguished name subject. Example: Austin
SSL_CLIENT_S_DN_O	Apache HTTPS	The organization of the client certificate distinguished name subject. Example: Microfocus
SSL_CLIENT_S_DN_OU	Apache HTTPS	The organization unit of the client certificate distinguished name subject. Example: R&D
SSL_CLIENT_S_DN_ST	Apache HTTPS	The state of the client certificate distinguished name subject. Example: TX
SSL_CLIENT_VERIFY	Apache HTTPS	Contains SUCCESS if the client verification was successful.
SSL_CLIENT_V_END	Apache HTTPS	The client certificate's validity end time. Example: Dec 16 20:27:44 2006 GMT
SSL_CLIENT_V_START	Apache HTTPS	The client certificate's validity start time. Example: Dec 16 20:17:44 2005 GMT
SSL_PROTOCOL	Apache HTTPS	The version of the SSL protocol. Example: SSLv3
SSL_SERVER_M_VERSION	Apache HTTPS	The server's certificate's version. Example: 1
SSL_SERVER_M_SERIAL	Apache HTTPS	The server's certificate's serial number. Example: 00
SSL_SERVER_S_DN	Apache HTTPS	The server certificate distinguished name subject. Example: /C=US/ ST=Texas/L=Austin/O=Micro Focus/OU=Development/ CN=cent32.microfocmi.com/ emailAddress=michael.schu ltz@microfocus.com
SSL_SERVER_S_DN_C	Apache HTTPS	The country of the server certificate distinguished name subject. Example: US
SSL_SERVER_S_DN_CN	Apache HTTPS	The computer name of the server certificate distinguished name subject. Example: cent32.microfocus.com
SSL_SERVER_S_DN_Email	Apache HTTPS	The email address of the server certificate distinguished name subject. Example: michael.schultz@microfocu s.com
SSL_SERVER_S_DN_L	Apache HTTPS	The location of the server certificate distinguished name subject. Example: Austin

Variable	Platform	Description
SSL_SERVER_S_DN_ST	Apache HTTPS	The state of the server certificate distinguished name subject. Example: Texas
SSL_SERVER_S_DN_O	Apache HTTPS	The organization of the server certificate distinguished name subject. Example: Micro Focus
SSL_SERVER_S_DN_OU	Apache HTTPS	The organization unit of the server certificate distinguished name subject. Example: Development
SSL_SERVER_I_DN	Apache HTTPS	The server certificate issuer's distinguished name subject. Example: /C=US/ST=Texas/ L=Austin/O=Micro Focus/ OU=Development/ CN=cent32.microfocus.com/ emailAddress=michael.schultz@microfocus.com
SSL_SERVER_I_DN_C	Apache HTTPS	The country of the server certificate issuer's distinguished name subject. Example: US
SSL_SERVER_I_DN_CN	Apache HTTPS	The computer name of the server certificate issuer's distinguished name subject. Example: cent32.microfocus.com
SSL_SERVER_I_DN_Email	Apache HTTPS	The email address of the server certificate issuer's distinguished name subject. Example: michael.schultz@microfocus.com
SSL_SERVER_I_DN_L	Apache HTTPS	The location of the server certificate issuer's distinguished name subject. Example: Austin
SSL_SERVER_I_DN_O	Apache HTTPS	The organization of the server certificate issuer's distinguished name subject. Example: Micro Focus
SSL_SERVER_I_DN_OU	Apache HTTPS	The organization unit of the server certificate issuer's distinguished name subject. Example: Development
SSL_SERVER_I_DN_ST	Apache HTTPS	The state of the server certificate issuer's distinguished name subject. Example: Texas
SSL_SERVER_A_KEY	Apache HTTPS	The signature algorithm of the server's key. Example: rsaEncryption
SSL_SERVER_A_SIG	Apache HTTPS	The signature algorithm of the server's certificate. Example: md5WithRSAEncryption

Variable	Platform	Description
SSL_SERVER_V_END	Apache HTTPS	The server certificate's validity end time. Example: `Jan 13 08:13:27 2006 GMT`
SSL_SERVER_V_START	Apache HTTPS	The server certificate's validity start time. Example: `Dec 14 08:13:27 2005 GMT`
SSL_VERSION_INTERFACE	Apache HTTPS	The version of the SSL interface. Example: `mod_ssl/2.0.55`
SSL_VERSION_LIBRARY	Apache HTTPS	The version of the SSL library. Example: `OpenSSL/0.9.8a`
UNMAPPED_REMOTE_USER	IIS	The name of the user as it is derived from the authorization header sent by the client, before the user name is mapped to a Windows account. This variable is no different from REMOTE_USER. If you have an authentication filter installed on your web server that maps incoming users to accounts, use LOGON_USER to view the mapped user name.
URL	IIS	Gives the base portion of the URL.

XML Exchange Request File Format

Here is a sample request, as written to the file specified by the BIS_FILENAME environment variable. The request is transmitted in XML and is wrapped in the following top-level element:

```
<?xml version="1.0" encoding="UTF-8" ?>
< bis:request xmlns:bis=http://www.xcentrisity.com/2003/bis/request >
    content, cookies, queryparams, server variables
</ bis:request >
```

The *content* , *cookies* , *queryparams* , *server variables* contains the four elements described in the following table:

`< bis:content >`
`  payload data`
`</ bis:content >`
	Contains the content part of the request (such as form variables POSTed back to the server). This element will be empty if there is no content data in the request-as is typically true of the first (GET) request.

`< bis:cookies >`
`  < bis:cookie name=name >`
`  cookie data`
`  < /bis:cookie >`
`  …`
`</ bis:cookies >`
	Contains an attributed <bis:cookie> element for each cookie that was transmitted with the request.

`< bis:query-params >`
`  < bis:query-param name=name >`
`  parameter data`
`  </ bis:query-param >`
`  …`
`</ bis:query-params >`
	Contains an attributed <bis:query-param> element for each query parameter that was transmitted with the request.

`< bis:server-variables >`
`  < bis:server-variable name=name >`
`  server variable data`
`  </ bis:server-variable >`
`  …`
`</ bis:server-variables >`
	Contains an attributed `<bis:server-variable>` element for each server variable associated with this request.

The content of a sample request file is below. Note that this is also visible in the trace output, if tracing is enabled. Also note that the `<bis:content>` section is application-dependent. This particular example is from the *http://localhost/xbisvcob/samples/MF/sample3* application with the following data entered into the form fields:

Element	Attribute	Value
`numberOne`		`5`
`numberTwo`		`2`
`cookie`	`BISKIT`	`Vos9tBgZknXRMTyI4GaJKw`

Because this is a web service sample, there are no form fields or query parameters to store into the `<bis:content>` and `<bis:query-params>` elements. However the cookies are stored as attributed elements into the `<bis:cookies>` section. Finally, all server variables are output into the `<bis:server-variables>` section (not depicted above). Using Xcentrisity XML Extensions and XSLT, the service program can selectively extract any or all of these elements and ignore elements that are not important to the application.

Here is the complete XML exchange file for this example. Note that the XML tags are indented to make the example easier to read.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<bis:request xmlns:bis="http://www.xcentrisity.com/2003/bis/request">
  <bis:content>
    <SOAP-ENV:Envelope
      xmlns=""
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:s="http://www.w3.org/2001/XMLSchema"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:tns="http://tempuri.org/bis/samples/Calculator/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <SOAP-ENV:Body>
        <Add xmlns="http://tempuri.org/bis/samples/Calculator/">
          <A>5</A>
          <B>2</B>
        </Add>
      </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
  </bis:content>
  <bis:cookies>
    <bis:cookie name="cookies">true</bis:cookie>
    <bis:cookie name="BISKIT">Vos9tBgZknXRMTyI4GaJKw</bis:cookie>
  </bis:cookies>
  <bis:query-params/>
  <bis:server-variables>
    <bis:server-variable name="BIS_ROOT_PATH">
/xbisvcob/samples
    </bis:server-variable>
    <bis:server-variable name="HTTP_ACCEPT">*/*</bis:server-variable>
    <bis:server-variable name="HTTP_ACCEPT_LANGUAGE">
en-us
```

```
    </bis:server-variable>
    <bis:server-variable name="HTTP_REFERER">
http://tex-mikes-centos54/xbisvcob/samples/sample3/
    </bis:server-variable>
    <bis:server-variable name="HTTP_SOAPACTION">
"http://tempuri.org/bis/samples/action/Calculator.Add"
    </bis:server-variable>
    <bis:server-variable name="CONTENT_TYPE">
text/xml; charset="UTF-8"
    </bis:server-variable>
    <bis:server-variable name="HTTP_ACCEPT_ENCODING">
gzip, deflate
    </bis:server-variable>
    <bis:server-variable name="HTTP_USER_AGENT">
 Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; WOW64; Trident/4.0;
GTB6.5; SLCC1; .NET CLR 2.0.50727; .NET CLR 3.5.30729; InfoPath.2;
OfficeLiveConnector.1.3; OfficeLivePatch.0.0; .NET CLR 1.1.4322; MS-RTC EA 2;
MS-RTC LM 8; .NET CLR 3.0.30729)
    </bis:server-variable>
    <bis:server-variable name="HTTP_HOST">
tex-mikes-centos54
    </bis:server-variable>
    <bis:server-variable name="CONTENT_LENGTH">613</bis:server-variable>
    <bis:server-variable name="HTTP_CONNECTION">
Keep-Alive
</bis:server-variable>
    <bis:server-variable name="HTTP_CACHE_CONTROL">
no-cache
    </bis:server-variable>
    <bis:server-variable name="HTTP_COOKIE">
cookies=true; BISKIT=Vos9tBgZknXRMTyI4GaJKw
    </bis:server-variable>
    <bis:server-variable name="PATH">
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/
bin:/usr/sbin:/usr/bin:/root/bin
    </bis:server-variable>
    <bis:server-variable name="SERVER_SIGNATURE"></bis:server-variable>
    <bis:server-variable name="SERVER_SOFTWARE">
Apache/2.2.14 (Unix) mod_ssl/2.2.14 OpenSSL/0.9.8l
    </bis:server-variable>
    <bis:server-variable name="SERVER_NAME">
tex-mikes-centos54
    </bis:server-variable>
    <bis:server-variable name="SERVER_ADDR">10.64.26.41</bis:server-variable>
    <bis:server-variable name="SERVER_PORT">80</bis:server-variable>
    <bis:server-variable name="REMOTE_ADDR">10.64.26.22</bis:server-variable>
    <bis:server-variable name="DOCUMENT_ROOT">
/usr/local/apache22/htdocs
    </bis:server-variable>
    <bis:server-variable name="SERVER_ADMIN">
michael.schultz@microfocus.com
    </bis:server-variable>
    <bis:server-variable name="SCRIPT_FILENAME">
/var/local/xbisvcob/samples/mf/sample3/calculator.srf
    </bis:server-variable>
    <bis:server-variable name="REMOTE_PORT">63994</bis:server-variable>
    <bis:server-variable name="GATEWAY_INTERFACE">CGI/1.1</bis:server-
variable>
    <bis:server-variable name="SERVER_PROTOCOL">HTTP/1.1</bis:server-variable>
    <bis:server-variable name="REQUEST_METHOD">POST</bis:server-variable>
    <bis:server-variable name="QUERY_STRING"></bis:server-variable>
    <bis:server-variable name="REQUEST_URI">
/xbisvcob/samples/sample3/calculator.srf
    </bis:server-variable>
```

```
    <bis:server-variable name="SCRIPT_NAME">
/xbisvcob/samples/sample3/calculator.srf
    </bis:server-variable>
    <bis:server-variable name="HTTP_URL">
/xbisvcob/samples/sample3/calculator.srf
    </bis:server-variable>
  </bis:server-variables>
</bis:request>
```

Windows/UNIX Portability Considerations

BIS is designed to allow web applications and services to be portable between Windows and UNIX-based web servers and operating systems. This means that, with some care, the developer can produce stencils (that is, `.srf` files) and service programs that do not depend on platform-specific features or characteristics and are, thus, portable. If a portable application is the goal, the following issues must be considered.

- The `Handler` tag is required for all platforms; however the parameter has no effect when rendered on UNIX. For portability, specify `{{ Handler * }}`.
- Pathnames referenced by stencils and service programs are subject to the differences between Windows and UNIX file naming conventions/rules. If portability is an objective, they must be chosen carefully. In particular, UNIX file naming is case-sensitive, and Windows is not. This means that a portable application should be consistent in its use of case within file names, and the files themselves should be named in accordance with that consistent use.

  If there is any possibility that a BIS application will be moved between UNIX and Windows, it is a good practice to restrict filenames to all lower-case names without any embedded spaces.
- Pathnames are also subject to the different conventions regarding the directory edge-name separator (`/` vs. `\`). In order to enable portable `.srf` files, BIS allows the `/` to be used on both Windows and UNIX everywhere except in the Handler tag. If portability is the goal, the `\` character should not be used as a pathname separator.
- There are a few features that are implemented in BIS/IIS on Windows, but have not yet been implemented on UNIX. These are called out with the IIS icon in the section of this document where the feature is described.
- Newer versions of BIS support tags that may not be recognized by older versions.

No application should be assumed to be portable unless it has been tested in every environment to which it is expected to be deployed.

Regular Expression Syntax

Regular expressions may be used in the `MATCH` and `SUBSTITUTE` parameters of the `Value` tag.

Metacharacters

This table lists the metacharacters that may be used in `{{Value(…MATCH= regexp )}}` and `{{Value(…SUBSTITUTE= regexp )}}`.

Metacharacter	Meaning
`.`	Matches any single character.
`[ ]`	Indicates a character class. Matches any character inside the brackets (for example, `[abc]` matches `a`, `b`, and `c`).

Metacharacter	Meaning
^	If this metacharacter occurs at the start of a character class, it negates the character class. A negated character class matches any character except those inside the brackets (for example, `[^abc]` matches all characters except a, b, and c). If ^ is at the beginning of the regular expression, it matches the beginning of the input (for example, `^[abc]` will only match input that begins with a, b, or c).
-	In a character class, indicates a range of characters (for example, `[0-9]` matches any of the digits 0 through 9).
?	Indicates that the preceding expression is optional: it matches once or not at all (for example, `[0-9][0-9]?` matches 2 and 12).
+	Indicates that the preceding expression matches one or more times (for example, `[0-9]+` matches 1, 13, 666, and so on).
*	Indicates that the preceding expression matches zero or more times.
??, +?, *?	Non-greedy versions of ?, +, and *. These operators match as little as possible, unlike the greedy versions which match as much as possible. Example: given the input <abc><def>, `<.*?>` matches <abc> while `<.*>` matches <abc><def>.
( )	Grouping operator. Example: `(\d+,)*\d+` matches a list of numbers separated by commas (such as 1 or 1,23,456).
{ }	Indicates a match group.
\	Escape character: interpret the next character literally (for example, `[0-9]+` matches one or more digits, but `[0-9]\+` matches a digit followed by a plus character). Also used for abbreviations (such as `\a` for any alphanumeric character; see the table below). If \ is followed by a number *n* , it matches the *n* th match group (starting from 0). Example: `<{.*?}>.*?</\0>` matches <head>Contents</head>.
$	At the end of a regular expression, this character matches the end of the input. Example: `[0-9]$` matches a digit at the end of the input.
|	Alternation operator: separates two expressions, exactly one of which matches (for example, `T|the` matches The or the).
!	Negation operator: the expression following ! does not match the input. Example: a!b matches a not followed by b.

Abbreviations

Abbreviations such as `\d` instead of `[0-9]` are allowed. The following abbreviations are recognized:

Abbreviation	Expansion	Matches
`\a`	`([a-zA-Z0-9])`	Any alphanumeric character
`\b`	`([ \t])`	White space (blank)
`\c`	`([a-zA-Z])`	Any alphabetic character
`\d`	`([0-9])`	Any decimal digit
`\h`	`([0-9a-fA-F])`	Any hexadecimal digit
`\n`	`(\r|(\r?\n))`	Newline (both Windows and UNIX)
`\q`	`(\"[^\"]*\")|(\'[^\']*\')`	A quoted string (either single or double quotes)
`\w`	`([a-zA-Z]+)`	A simple word
`\z`	`([0-9]+)`	An integer

BIS Troubleshooting Tips

This Appendix outlines the symptoms of some common abnormal conditions, and provides insight as to the possible cause(s) and corrective action(s).

Before troubleshooting, if you are using Internet Explorer, be sure that the **Show Friendly HTTP error messages** option is not checked. This option can be found in **Tools > Internet Options > Advanced > Browsing** in either Internet Explorer or **Control Panel > Internet Options**.

- **Symptom:**

```
Server Error in Application "Default Web Site/xbisvcob"

HTTP Error 500.0 - Internal Server Error
Description: Handler "AboMapperCustom-24582078" has a bad module
     "IsapiModule" in its module list
Error Code: 0x8007000d
Notification: ExecuteRequestHandler
Module: IIS Web Core

Requested URL: http://localhost:80/xbisvcob/samples/MF/default.srf?
trace=page
Physical Path: C:\inetpub\wwwroot\xbisvcob\samples\MF\default.srf
Logon User: Anonymous
Logon Method: Anonymous
Handler: AboMapperCustom-24582078
```

- **Possible Cause:** Indicates that IIS ISAPI extension support is not installed.
- **Suggestion:** In Windows 2008 Server, start **Programs and Features** in the Windows control panel, and ensure that **ISAPI Extensions** are enabled (that is, the option is checked) under **Internet Information Services > World Wide Web Services > Application Development Features**.
- **Symptom:**

```
Xcentrisity Business Information Server Error
An error occurred while BIS was processing your request. Additional
information is below.
XMLExchange failed: the service program returned error
"80004004", which is "Operation aborted". The session has ended.
```

- **Possible Cause:** Indicates that there was a problem starting the Service Engine.
- **Suggestion:** To narrow the problem, turn on tracing by adding this tag to your `.srf` file:

  ```
  {{ Trace(start, page) }}
  ```

  Then, refresh the page. You should now see a table headed `Request Details` at the end of the page. Scroll down to `Trace Information` and look for `Service` in the left-most column.

  The BIS samples are pre-configured for tracing and tracing may be turned on and off with a query parameter defined in the `Trace` tag. For example, if the problem occurred running the `VERIFYBIS` program, log into the server running BIS and use this URL:

  ```
  http://localhost/xbisvcob/samples/MF/verify/default.srf?trace=page
  ```

  Trace output will appear at the bottom of the page, and this will include the BIS Service Engine startup messages that should reveal the problem.
- **Symptom:** An error 500 occurs.
- **Possible Cause:** A replacement tag precedes the Handler tag.
- **Suggestion:** The only tags allowed before the Handler tag are comment tags. Move all tags that precede the Handler tag to follow it.
- **Symptom:** One of the following error messages is reported:

  ```
  Cannot create the trace file for session "nH6shZykCtbZmdDZHo0LhJhiVSq5"
  (the last attempted filename is "D:\Documents and Settings\UserID\Local
  Settings\Temp\BIS-nH6s-trace.txt"). The last error code was 80070005

  Cannot reopen the trace file for session "nH6shZykCtbZmdDZHo0LhJhiVSq5"
  (the last attempted filename is "D:\ Documents and Settings\UserID\Local
  Settings\Temp\BIS-nH6s-trace.txt"). The last error code was 80070005

  Could not write the trace file to the directory "D:\ Documents and Settings
  \UserID\Local Settings \Temp\": the error code was 80004005.
  ```
- **Suggestion:** To correct this error, give the IWAM_* account write access to this directory. See the *Troubleshooting* appendix in the *User's Guide* for more information.
- **Symptom:** The following error message appears in the web browser:

  ```
  Server Error
  LoadLibrary failed.
  ```
- **Possible Cause:** The Handler tag is missing, invalid, or refers to a missing or invalid library.
- **Suggestion:** Make sure that your `.srf` page has a `{{ Handler * }}` tag, and that this tag is the first non-comment/non-include tag in the file. For Windows, it must also appear in the first 4096 characters of the `.srf` file.

Configuring BIS/IIS after Installation

The Business Information Server Service Engine must be registered with Windows. If it becomes necessary to re-register the server, registration can be performed:

- by reinstalling BIS/IIS (choose the **Repair** option), or
- from the command line

This section describes how to configure the BIS/IIS Service Engine from the command line.

Command Line Configuration

BIS is self-registering. Registration is performed by the XBIS.EXE program, which can be found in the installation directory (normally `C:\Program Files\Micro Focus\Xcentrisity BIS for Visual COBOL`). Registration includes these three steps:

- The BIS Service Engine contained in XBIS.EXE is registered.
- The Run As identity, that is, the identity that will be used to execute service programs, is set.

The server registration syntax is:

```
XBIS registration-options
```

The registration options are detailed below:

/REGSERVER	Registers the Service Engine. Also registers the runtime system located in the same directory as XBIS.EXE.
/UNREGSERVER	Unregisters the BIS Service Engine and the runtime system.
/SHOWSERVER	Displays a dialog box that shows the location of the currently registered BIS and Service Engine.

The server registration option has three additional variations:

/REGSERVERQ	Quietly registers the BIS Service Engine and the runtime system located in the same directory as XBIS.EXE. No confirmation dialog box is displayed .
/REGSERVERO	Only registers the BIS Service Engine. The runtime system registration remains unchanged. This is useful if you want to install the runtime system in a directory separate from the BIS Service Engine.
/REGSERVERQO	Combines the above two options.

The /REGSERVER and /REGSERVERQ options have an additional optional parameter: the pathname of the runtime system or the directory containing the runtime system. It is specified as in these examples:

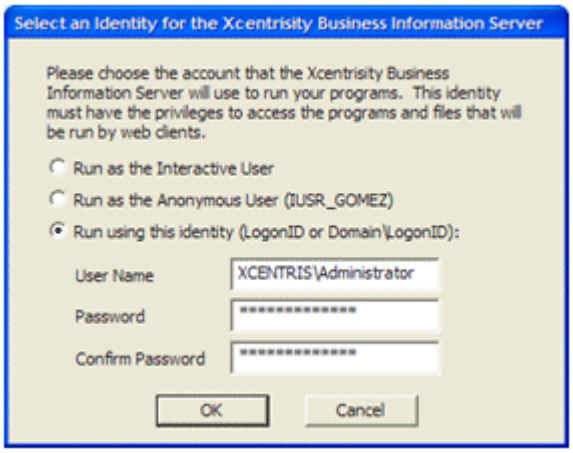
```
1. /REGSERVER:pathname
2. /REGSERVERQ:pathname
3. /REGSERVER:directory
4. /REGSERVERQ:directory
```

If the pathname or directory is specified, the specified file or the server in the specified directory is registered and BIS does not search for the runtime system in the path.

If a directory is specified, it may end with a trailing backslash to differentiate it from a filename. Also note that if the specified name contains spaces, it must be surrounded by single or double quotes.

Configuring the Run As Logon ID

To execute service programs, Business Information Server must assume the identity of a user authorized to run the programs and access data files required by the programs. This is accomplished by specifying a Logon ID during installation, reinstallation, or server registration.



The *Run As* identity may be configured during registration interactively with a dialog box, or by specifying options on the command line.

Note that the /RUNAS options below must be specified along with one of the /REGSERVER options described above.

If none of the options in the table below are specified, the server displays the Run As configuration dialog box on the right even if /REGSERVERQ (*quiet mode*) is specified.

The **Run As** dialog box has three options that determine the context in which BIS will execute:

/RUNASI /RUNASIP	Causes the server to run as the INTERACTIVE USER. This is the identity of the user that is logged on to the server's console. This is most useful for developers but is not recommended for deployment. If the P suffix is specified, BIS prompts for credentials using the dialog box if an error occurs.
/RUNASL /RUNASLP	Runs the server under the identity of the launching (usually anonymous) user. This will normally be the account named IUSR_*machinename*, where *machinename* is the name assigned to the machine. For example, if your machine is named HILO, the anonymous user's name is IUSR_HILO. It is possible for a system administrator to change this, either for all IIS accounts or for just the BIS. If the name of the machine was changed after IIS was installed, this will be the original name of the machine, not the current name. In this case, please see *Manual Configuration*, below. Note that this account usually has very limited privileges and BIS will not even be able to start unless you manually give this account write permission in the BIS installation directory. BIS will not be able to access files in other directories, unless you also give it access to those directories, and will not be able to access files on any network volumes unless your machine is joined to a domain and this name is known to the domain server. See your system administrator for details. If the P suffix is specified, BIS prompts for credentials using the dialog box if an error occurs.

/RUNAS: *id, pw*
/RUNASP: *id, pw*
	Runs the server using the specified identity. This is the recommended option. id is the login ID and *pw* is the password. The password is encrypted by Windows, is stored in the registry, and is not retrievable as plain text once the server is registered. However, caution is required when embedding a clear-text password in a batch file that issues the /RUNAS command.
	If an id is specified without a *pw*, the program prompts for the password. This may be a good compromise between convenience and security.
	Either the *pw* id or the *pw* may be quoted with single or double quotes (required if either contains spaces). The entire parameter string may also be quoted.
	Examples:
	`/RUNAS:myuserid, mypassword`
	`/RUNAS:"my user id","my password"`

```
/RUNAS:"my user id,my password"
/RUNAS:"INTERACTIVE USER"
```

As a special case, the special logon ID of `INTERACTIVE USER` is recognized and handled as if `/RUNASI` were specified. Any password is ignored, and quotes are required due to the embedded space.

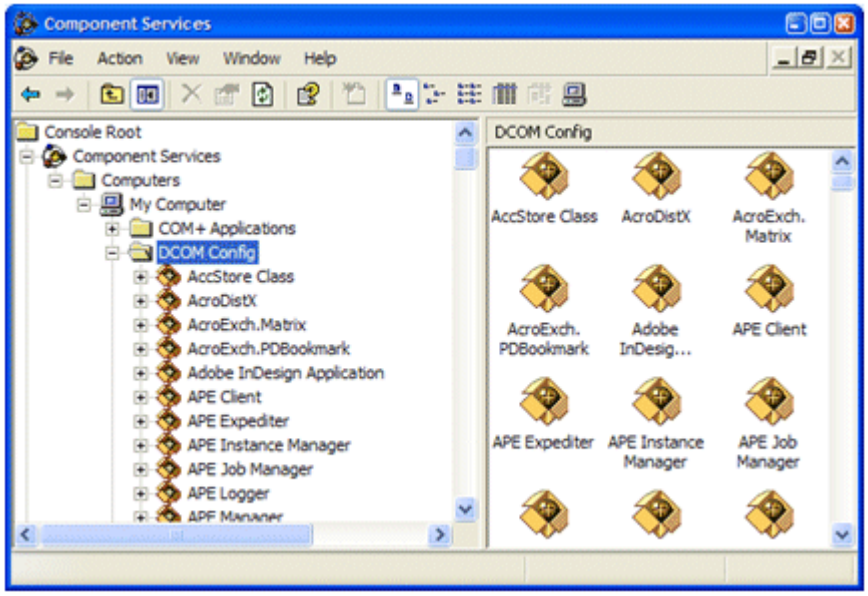
If the P suffix is specified, BIS prompts for credentials using the dialog box if an error occurs.

Retrieving or Changing the Configured Identity

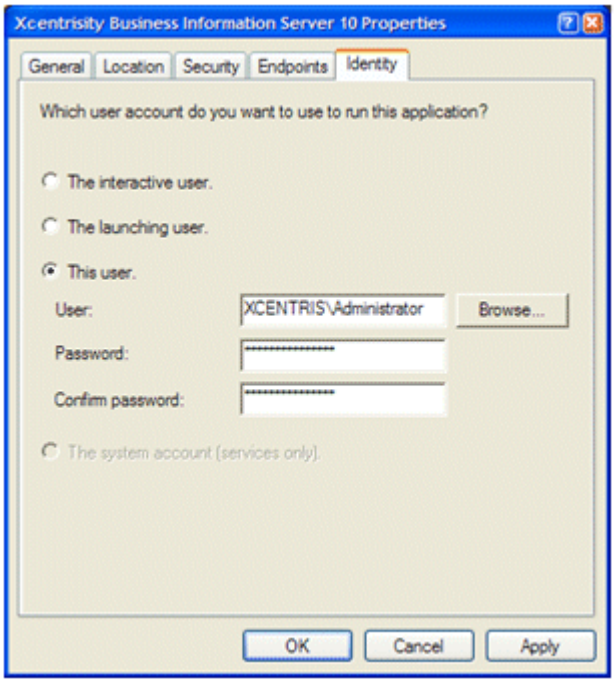
The Windows Component Services configuration utility may be used to examine and change the current Business Information Server configuration.

There are two ways to start the utility:

- Select **Start > Control Panel > Administrative Tools > Component Services**. (Alternatively, select **Start > Run**, enter `dcomcnfg` in the **Open** box, and click the **OK** button.)
- Select **Start > Control Panel > Administrative Tools > Component Services**. The program should look like this:



1. Find **Xcentrisity Business Information Server *xx*** in the list, right-click, and select **Properties** from the popup menu.
2. Click the **Identity** tab. The dialog box depicted below displays the current `Run As` configuration.



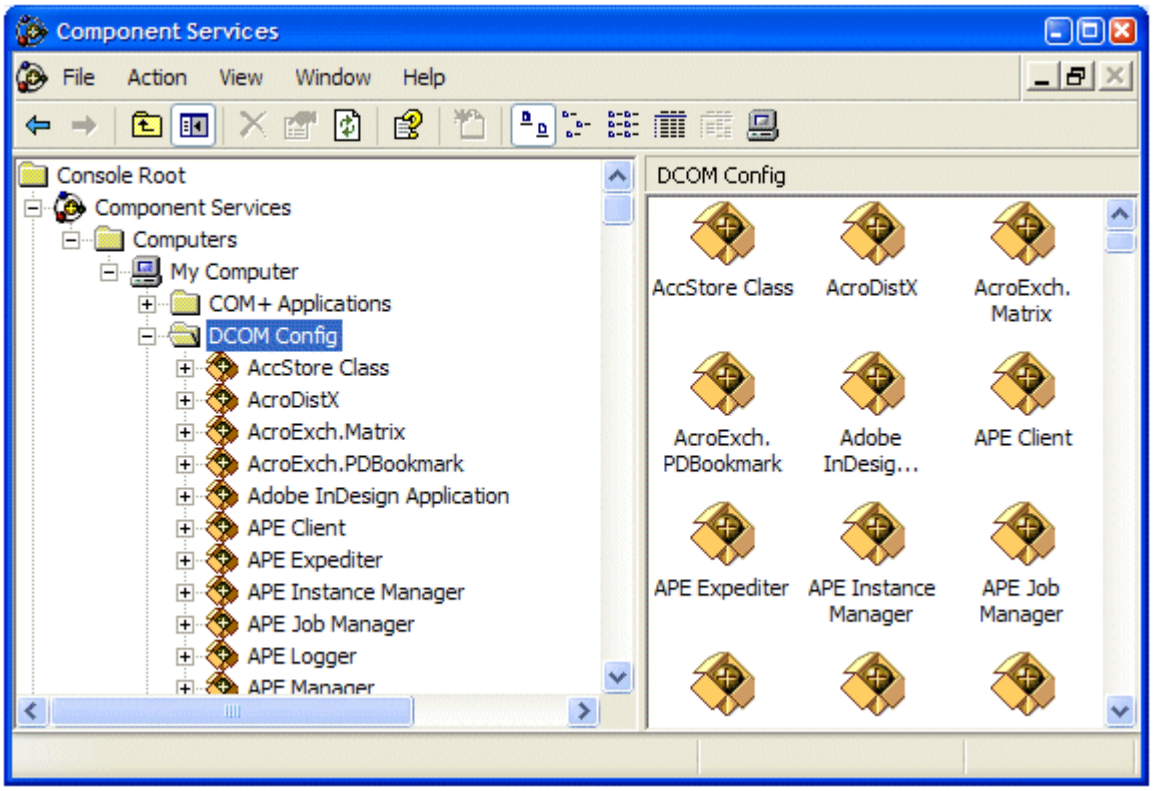
Note that you can change the identity and/or the password that BIS/IIS uses to run service programs here.

Manual Configuration

To manually change the user ID and password that the Service Engine uses to execute programs, follow these steps after completing the installation:

1. Select **Start > Control Panel > Administrative Tools > Component Services**.

   Alternatively, select **Start > Run**, enter `dcomcnfg` in the **Open** box, and click the **OK** button.
2. Expand **Console Root > Component Services > My Computer > DCOM Config**. The program should look like this:



3. Locate **Xcentrisity Business Information Server *xx*** in the list, right-click, and select **Properties** from the popup menu.
4. Click the **Identity** tab, then **This user**. Enter the user ID and the password that you want to use to run service programs under **Business Information Server**. Then click the **Apply** button.
5. Click the **Security** tab and under **Launch Permissions**, click **Customize** and then click **Edit**. Click **Add** and enter the name of your anonymous internet account (see below). Click the **Add** button; make sure Allow is checked next to Launch Permission and click **OK**. Then click **Apply**.
6. Still on the **Security** tab, repeat the above step for **Access Permissions**.
7. You do not need to change **Configuration Permissions**. Click **OK** to close the dialog box.

The name of your anonymous internet account is normally `IUSR_`*`machine`*, where *`machine`* is the hostname assigned to your machine. However, the system administrator can change the name of this account, and this is common if you are running more than one web site.

To determine the name of your anonymous internet account:

1. Select **Start > Control Panel > Administrative Tools > Internet Information Services**.
2. Expand **Internet Information Services > Local Computer > Web Sites > Default Web Site**. (Replace the last node with your site if IIS is serving multiple web sites).
3. Find the virtual directory that was created to contain the BIS service program. This will be xbisvcob for the sample program. Right-click that node and select **Properties**.
4. Click **Directory Security**, then **Edit**.
5. The **User Name** box contains the name of the anonymous account that you can enter above.

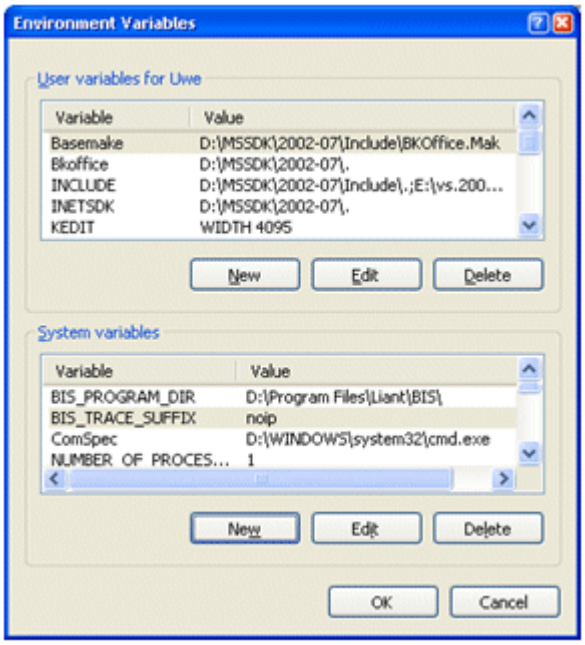
Note that the above configuration is very flexible. You can control what users will have access to the COBOL program on a site-by-site, or even a directory-by-directory basis on your web site.

Alternatively, instead of specifying `IUSR_`*`machine`*, you can specify `GUEST`, or any other group that contains all your anonymous access accounts. However, be cautious before granting too many privileges to too many anonymous processes.

Setting Environment Variables

Some BIS settings are set from the server environment. To set a BIS environment variable:

- Log in as `Administrator`, or an account that is a member of the Administrators group.
- Click **Start > Control Panel > System**.
- Click the **Advanced** tab.
- Click the **Environment Variables** button.
- Under **System Variables**, click the **New** button. Alternatively, if the environment variable has already been set, click the variable name in the list box and then click the **Edit** button.
- Enter the variable name and the value and select **OK**.
- When done, click **OK** to dismiss the dialog box.



The changes take effect immediately.

Setting the Maximum Thread Count

BIS uses a system resource called a Thread to render pages. For efficiency, BIS maintains an internal pool of threads, and when a request for a BIS page arrives, a thread from the pool is dispatched to serve the page. When the page is completely rendered, the thread returns to the pool to await the next request.

If there are no available threads in the pool, the request must wait for a thread to become available. A request will wait for some period of time (normally about 60 seconds) before being denied with a `server too busy` error page.

BIS pages that do not communicate with the Service Engine normally execute very quickly. However, if a page contains an `XMLExchange` tag, the BIS thread serving that page must wait until the Service Engine provides the replacement text for the `XMLExchange` tag. If this is a lengthy process, it is conceivable that BIS will not have enough threads to serve all pending requests. In this case, it may be desirable to increase the size of the BIS thread pool so more pages can be rendered simultaneously.

The `BIS_MAX_THREADS` environment variable may be used to increase (or decrease) the size of the thread pool. The syntax is:

```
BIS_MAX_THREADS=value
```

where:

n	Is an integer that specifies the number of threads that will be used by BIS to service requests.

Notes

- Since each BIS thread requires system resources, even when idle, it is not desirable to set this value to a large number. The default value, 5 threads, is sufficient for a moderately busy server and should only be increased if requests are being denied or users are waiting for their requests to be serviced.
- BIS dynamically creates additional threads for each Service Engine started by the `StartService` tag. These Service Engine threads do not count against the `BIS_MAX_THREADS` value.
- The `BIS_MAX_THREADS` option is only examined when the BIS Request Handler is loaded. The handler is loaded on demand, for example, when the first BIS request arrives after a server restart, and then the handler is automatically unloaded after about 20 minutes of inactivity.
- The current setting can be retrieved with `Value(MaxThreads, Config)`. On UNIX, this always returns 1.

Configuration after Installation (UNIX/Apache)

Configuring Apache

The Apache configuration file for BIS is named `mod_xbis.conf` and is included in the Apache server configuration by an Include directive placed in the main `httpd.conf` configuration file. This shows the Include directive.

If available, copy or link the `mod_xbis.conf` file to the `/etc/httpd/conf.d` directory. This circumvents the necessity of editing the main `httpd.conf` configuration file.

The BIS Configuration File

The BIS configuration file contains several sets of Apache configuration directives. The first set of directives configures Apache direct requests to the BIS Request Handler module. This sample shows this set of directives.

```
LoadFile                /opt/microfocus/VisualCOBOL/lib/libxml.so
LoadModule xbis_module  /opt/microfocus/VisualCOBOL/lib/mod_xbis22.so
AddHandler bis-stencil  srf
AddType    text/html    srf
AddType    text/x-component .htc
```

The LoadFile directive is required and should not be changed. It causes Apache to dynamically load the shared object containing the BIS Request Handler's XML parser when Apache starts.

The LoadModule directive is required and should not be changed. It causes Apache to dynamically load the shared object containing the BIS Request Handler when Apache starts.

The AddHandler directive causes all URIs that request files ending with `srf` to be processed by the BIS Request Handler. If it is desired to have the Request Handler process requests with other file extensions, add additional AddHandler directives.

The AddType directive causes the default content type of a response for a URI ending with `srf` to be text/html. An AddType directive should be added for each AddHandler directive added to serve an addition file extension.

The AddType directive for the `.htc` extension is necessary to cause Apache to serve HTML Components files (a Microsoft extension) with the correct content type.

These directives affect the amount and location of trace information produced by BIS.

```
BISTraceDirectory /var/xbis
BISTraceFile trace.log
BISKeepTraceFiles Off
BISTruncateTraceFile Off
BISTraceSuffix Page
BISMasterTrace On
BISMainDebug On
BISStencilDebug On
BISSEDebugLevel 0
```

The BISTraceDirectory directive specifies the directory where trace files are written. The default is `/var/xbis`. If this directive does not specify an absolute path, it is assumed to be relative to a default (`/tmp` on some systems).

The BISTraceFile directive indicates the name of the trace file. This directive should only be used when all tracing for all requests are written to the same file. If this directive does not specify an absolute path, it is relative to the directory specified by BISTraceDirectory.

The BISKeepTraceFiles directive controls whether trace files are kept after a session completes. The value of `Off` is the default, and it causes trace files to be deleted, unless a FILE option in a Trace tag (in a stencil file) requests that they be kept. The value of `On` causes trace files to be retained regardless of the presence of a FILE trace option.

The BISTruncateTraceFile directive controls whether trace files are truncated at the beginning of each request. The value of `Off` is the default and causes all requests of a session to be placed in the trace file. The value of `On` cause only the last request of the session to be placed in the trace file.

The BISTraceSuffix directive adds additional options to Trace tags (in a stencil file) whenever one is processed. The value of this directive is processed after the options specified in the Trace tag, but before the options specified in the trace query parameter. There is no default for this directive. The options are described in the Trace tag section. All Trace tag options are allowed.

The BISMasterTrace directive is a master switch that controls all tracing activity. The value of Off is the default and will prevent all tracing. This is the appropriate value for a production environment. The value of On allows tracing to occur.

The BISMainDebug directive controls tracing of tags as they are executed. The value of `Off` is the default and prevents trace messages. The value of `On` allows trace messages during execution of the stencil. This tracing approximates the tracing performed by BIS/IIS.

The BISStencilDebug directive controls tracing tags as they are parsed. The value of `Off` is the default and prevents trace messages. The value of `On` will cause trace messages diagnosing syntax errors in tags to be produced.

The BISSEDebugLevel directive controls tracing of the BIS Service Engine. The values are `0`, `1`, and `2`. `0` is the normal level of tracing and is appropriate for seeing DISPLAY statements from the service program. `1` and `2` supply additional tracing and should only be used when directed by customer support.

```
BISRefreshDirectory /var/tmp/xbis.refresh
```

The BISRefreshDirectory directive names a directory where server responses are stored temporarily, in case the client user agents such as web browsers request a refresh (see the XMLExchange tag.) The indicated directory should have permissions which allow create, reading, write, and delete access by the Apache child process. If no directory is named, or if this directive is omitted, the BIS Request Handler will not attempt to provide correct responses to refresh requests which lead to unnecessary session sequence errors.

```
BISErrorMessage ErrorName Error Text
```

The BISErrorMessage directive overrides the text for one of the BIS Request Handler's error messages. One reason to do this is to provide error messages in a language other than English. The first operand of the directive is the name of the error to be overridden. The remainder of the directive is the new text to be

displayed when ErrorName is encountered. The current set of the Request handler's error names and their text are present within `mod_xbis.conf` as commented out BISErrorMessage directives.

```
BISSesDaemonKey xxxxxxxx
```

The optional BISSesDaemonKey directive allows the shared memory key with which to contact the Service Engine to be specified. This directive should only be used when it is desired to run multiple Service Engine daemons on the same UNIX server. This is rare. The value is an 8-hex digit value that must match the SharedMemory option keyword of the configuration of the Service Engine to use.

```
Alias URL-Path Directory-Path
```

This standard Apache directive allows stencils (as well as other documents) to be served from directories outside of the Apache web server's document root. The *URL-Path* value is a string that is to be matched to the leading part of the path of desired URLs. When a match occurs, it is removed and replaced with the *Directory-Path* value to produce the actual file name of the requested document. When an Alias directive is used, create a corresponding Directory directive to specify additional configuration directives for the directory named `Directory-Path`.

```
<Directory Directory-Path>
    SetEnv BIS_ROOT_PATH /xbisvc22/samples
    SetEnv TEMP /var/xbis
    DirectoryIndex default.srf
</Directory>
```

This set of standard Apache directives demonstrates tailoring Apache directives to document directories. The *Directory-Path* value is the name of the directory to which the directives apply.

The SetEnv directives demonstrate setting server environment variables. The value of such a variable is available in a stencil in a *Value* tag. It is also available by enclosing its name between "%" characters. In the above example, %TEMP% in a stencil served from this directory would be replaced by `/var/xbis`.

The DirectoryIndex directive specifies the name of the default document to serve if only the directory name is specified in the requested URL.

Service Engine Configuration

The BIS Service Engine runs as a UNIX daemon process and one or more service processes which the daemon creates, as needed. There are always one or more idle service processes waiting for the Request Handler (the Apache part) to process a `StartService` tag.

Because the Service Engine runs as daemon, it normally starts when the operating starts, without any direct user interaction. It gets all of its options from a configuration file, its command line and its environment. The configuration file is usually named `/etc/xbis.conf`, but this can be changed by the `-f` command-line option. Each line in the configuration file is either a blank line, comment line or an option line. A comment line is a line in which the first nonblank character is a # character. On an option line, the line begins with a keyword, which is followed by one or more spaces or tabs and then by the option value. A # character may follow the option value to introduce an in-line comment.

The configuration file option keywords are:

`BinDir`	Specifies the name of the directory where the BIS binary executable files are located. There is no reason for a user to alter this parameter after installation.
`LibDir`	Specifies a colon separated list of directory names that will be placed into the standard search path environment variable for the UNIX platform when the Service Engine is started. Usually the environment variable is `LD_LIBRARY_PATH`, but it is `LIBPATH for AIX`, `SHLIB_PATH` for 32-bit HP-UX, and

	LD_LIBRARY_PATH_64 for 64-bit Solaris. The value of this option is prepended to the current value of the library search environment variable. There is no reason for a user to alter this parameter after installation.
LogDir	Specifies the name of the directory where the BIS log files are placed.
MaxChildren	Specifies the maximum number of service (child) processes. This is normally set to 250.
MaxSessions	Specifies the maximum number of BIS sessions. It defaults to twice the MaxChildren value.
PageSize	Specifies the amount of space allocated in the Sessions file for each session. This holds the information about a session between requests. It must be a power of two and it must be at least 512 but no more than 16384. It defaults to 2048, which should be sufficient unless your stencils define unusually long paths or a large number of environment variables.
SaveFiles	If specified, copies of all request and response files are saved in the temporary directory. This is a debugging tool, typically used during development of a web site.
ServiceTimeout	Default service timeout, in seconds. This is the preferred way to set the default service timeout. If BIS_SERVICE_TIMEOUT is set in the Apache configuration file for BIS (bis.conf), the Request Handler uses that value to override the value of the -T option. Doing so delays the start of each service program slightly.
SharedMemory	If present, specifies the shared memory key that the Service Engine is to use. This directive should only be used when it is desired to run multiple Service Engine daemons on the same UNIX server. The value is an 8-hex digit value that must be matched by the value BISSesDaemonKey directive in use by the Request Handler. Only specify this keyword option when directed by Micro Focus Technical Support.
Socket	Specifies the name of the socket used by the Request Handler to communicate with the Service Engine daemon. There is no reason for a user to alter this parameter after installation.
TempDir	Specifies the name of the directory where temporary files are created.
UserName	Specifies the UNIX user name used by each service (child) processes. Although the Service Engine daemon process runs as root, each of the child service processes runs as the user specified by this option. This determines the files that a service process can read and write, as well as the home directory of each service process.

Options on the Service Engine daemon's command line may modify the configuration as determined by the configuration file and the built-in defaults. The command-line options are in a string that is assigned to an

environment variable named `OPTIONS`. All of the Service Engine's environment variables, including `OPTIONS`, are set in a file named `/etc/sysconfig/xbis`. This file is created during the install of BIS.

The command-line options are:

`-f` *`file`*	Specifies the name of the Service Engine configuration file. If this option is present, it must be the first option on the command line. If omitted, the configuration file name defaults to `/etc/xbis.conf`.
`-c` *`count`*	Specifies the maximum number of service (child) processes. This is normally set to `9999` to indicate that the number of service processes is limited only by the license, but it may be set to a smaller value as a *throttle*.
`-i` *`count`*	Specifies the number of idle service (child) processes. This is normally set to `1` but a small increase in this may improve response time on a server which receives many requests in rapid succession.
`-T` *`timeout`*	Default service timeout, in seconds. This is the preferred way to set the default service timeout. If `BIS_SERVICE_TIMEOUT` is set in the Apache configuration file for BIS (`bis.conf`), the Request Handler uses that value to override the value of the `-T` option. Doing so delays the start of each service program slightly.
`-u` *`user`*	Specifies the UNIX user name used by each service (child) processes. Although the Service Engine daemon process runs as `root`, each of the child service processes runs as the user specified by this option. This determines the files that a service process can read and write, as well as the home directory of each service process.
`-t` *`dir`*	Specifies the name of the directory where temporary files are created.
`-r`	If specified, copies of all request and response files are saved in the temporary directory. This is a debugging tool, typically used during development of a web site.
`-L` *`file`*	Specifies the name of the Service Engine event log file. The Service Engine records certain important events in this file. This is a debugging tool.
`-s` *`file`*	Specifies the name of the socket used by the Request Handler to communicate with the Service Engine daemon. There is no reason for a user to alter this parameter after installation.
`-U` *`file`*	Specifies the name of a file used by the Service Engine daemon to communicate with the Request Handler. There is no reason for a user to alter this parameter after installation.

If the BIS Service Engine options need to be changed, the configuration file (/etc/xbis.conf) may be edited or (on systems other than AIX) the file /etc/sysconfig/xbis may be edited. If the configuration file is changed, the Service Engine can be instructed to reread it by using a kill command to send the Service Engine daemon a SIGHUP signal. However, the Service Engine does not read /etc/sysconfig/xbis directly. Instead, the shell script which starts the Service Engine reads this file. For any changes to take effect, the Service Engine must be restarted, either by restarting the operating system, by changing the runlevel, or by executing the shell script which starts the Service Engine (/etc/init.d/xbisengd). This script accepts one parameter, which must be one of the following:

Start	Starts the BIS Service Engine.
Stop	Stops the BIS Service Engine.
Restart	Stops the BIS Service Engine, and then starts it again.
Condrestart	If the Service Engine is running, stop it, and then start it again. Otherwise, do nothing.
Status	Displays the status of the Service Engine.

Note that stopping the Service Engine stops all of the service processes immediately, terminating any running service programs. This should not be used when users are connected to the server.

xbisctl Utility

The xbisctl utility can be used by a root user to control the Service Engine and the BIS Session/Logging daemon. It can also display the BIS sessions and, if necessary terminate a session. The xbisctl utility may be copied or linked to a directory in the user's path; it is located in the bin subdirectory of the directory where BIS was installed.

The xbisctl utility may be run in one of two ways. If no parameters are specified on the command line, it reads commands from standard input. Alternatively, a single command may be specified on the command line. The following table lists the commands that xbisctl recognizes:

Start	Starts the Service Engine and the Session/Logging daemon.
Stop	Stops the Service Engine and the Session/Logging daemon.
Status	Displays a one-line status for Service Engine and the Session/Logging daemon.
Refresh	Refreshes the Service Engine and the Session/Logging daemon. This tells the BIS daemons to reread their configuration file.
Sessions	List the current sessions.
Kill	Terminate a session.
Exit	Stop reading standard input. Alternatively, press **ctrl-D** to end input.

Status information can be displayed in a browser window. At the end of the supplied mod_xbis.conf file, there are two ScriptAlias directives. Uncomment one or both of these to enable this feature. The path may be changed to suit your needs. These run a shell script that executes the xbisctl utility with the status command on the command line.

Creating a BIS/IIS Virtual Directory

You can use the `BISMkDir` program to create and configure a virtual directory that is ready to run a BIS application. The `BISMkDir` program is installed in:

```
C:\Program Files\Micro Focus\Xcentrisity BIS for Visual COBOL
```

This program can also be downloaded from the Micro Focus SupportLine web site or obtained from Micro Focus SupportLine.

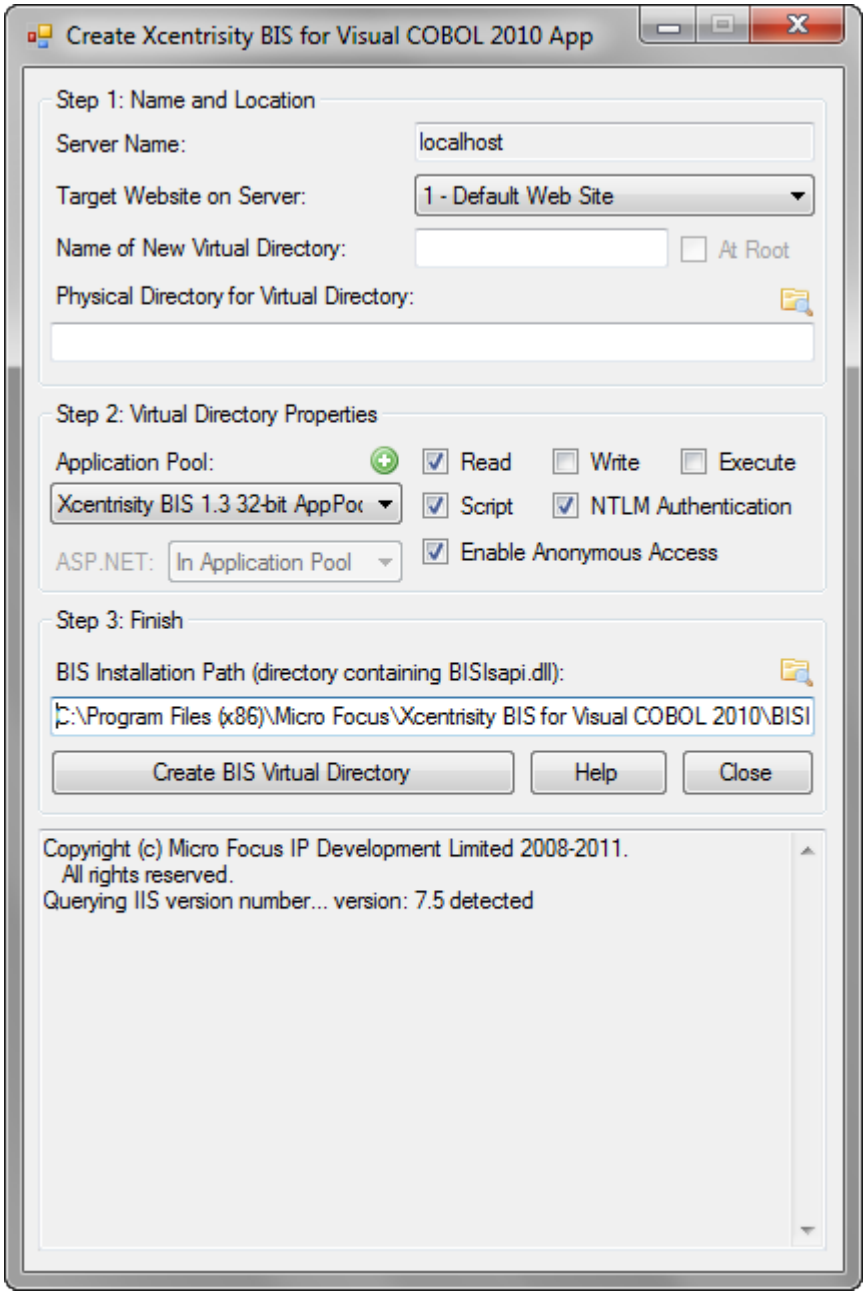
Running the BISMkDir Program

To run this program, from Windows Explorer or the command line, navigate to the following directory and either double-click the **BISMkDir** icon or run BISMkDir.exe.

```
C:\Program Files\Micro Focus\Xcentrisity BIS for Visual COBOL
```

On 64-bit Windows when a 32-bit-only system is installed, replace **Program Files** with **Program Files (x86)**.

When execution begins, you will see the following dialog box:



This dialog box has the following fields:

- **Server Name**

  In this release, always contains localhost. Note that this program currently has to be run on the system that contains the IIS server.
- **Target Website on Server**

  Select the website on the server that will serve the new virtual directory.
- **Virtual Root Name**

  Enter the name of the virtual directory that you wish created. For example, the default installation creates a virtual directory named xbisvcob.

  Note that, in this version of **BISMkDir**, the **At Root** checkbox is always disabled.
- **Physical Directory for Virtual Directory**

  Enter the pathname of the physical directory that will contain the files that are served when the user issues requests against the Virtual Directory Name.

For example, when BIS is installed in the default way and you request this page:

```
http://localhost/xbisvcob/samples/MF/default.srf
```

The requested content is served from:

```
C:\inetpub\wwwroot\xbisvcob\samples\MF\default.srf
```

This is because the BIS installer creates a physical directory named xbisvcob in the default web tree, and copies the sample programs into this directory. The installer then creates a virtual root directory named xbisvcob, configures it so it runs a BIS application (see below) and points it at the previously created physical directory.

Notes:

- The physical directory is not created if it does not exist.
- The physical directory must also have the appropriate permissions (for example, anonymous user read access) or BIS will not be able to serve files from this directory.
- It is usually convenient to create the physical directory in the web tree (for example, `c:\inetpub\wwwroot`) because the physical directory will inherit the permissions from the IIS parent directory. Otherwise, IIS will only manage the virtual directory permissions (read, write, execute), and the physical directory permissions must be separately managed.
- You may use the **Browse** button to browse for the directory.

- **Application Isolation Mode**

  The content of this drop-down list depends on the version of Windows that you are running. For IIS version 6, you may choose from the following options:

  - `0` - In-Process (low isolation)
  - `1` - Out-of-Process (high isolation)
  - `2` - Pooled (medium isolation)

  In-process means that the BIS request handler will run within the IIS process, along with all other in-process applications. This results in the best performance, but applications can interfere with each other, and if an application crashes, all applications are affected.

  Out-of-process runs each application in a separate process. Each BIS virtual directory will run in a separate process; BIS cannot interfere with other applications and other application failures will not affect BIS. This is the recommended isolation mode for BIS development.

  Pooled runs all applications designated as *pooled* together. Pooled applications can only interfere with other applications in the same pool. This is a compromise between the efficiency of in-process applications and the safety and reliability of out-of-process applications.

  Note that IIS 7 eliminates the in-process and out-of-process options and provides multiple application pools. This option is described below.

- **Application Pool**

  The content of this drop-down list depends on the version of Windows that you are running. For versions of IIS that support application pools, this drop-down contains a list of application pools that were found on the server. The name of the application pool will be suffixed with `(32-bit)` or `(64-bit)`.

  Note that currently BIS only supports 32-bit application pools. If the host is running a 64-bit version of Windows and no 32-bit application pool is found, a warning is issued and the virtual directory cannot be created until a 32-bit application pool is created.

- **Checkboxes**

  The checkboxes control how the virtual directory is created.

  - `Read` determines if web clients will have read permission to this virtual directory. This must be checked if BIS programs will be run in this directory.

- `Write` determines if web clients will be able to write to this virtual directory.

  Note: This should be enabled only for special purposes, as it is a security risk.
- `Execute` determines if programs can be executed in this virtual directory. This should not be enabled unless you are also using this directory as a CGI-type directory and plan to run programs out of this virtual directory on the web server.
- `Script` determines if scripts can be executed in this virtual directory. This must be checked if BIS programs will be run in this directory.
- `NTLM Authentication` should be checked to use this kind of authentication in this directory. In general, this box should be checked.
- **BIS Installation Path**

  This is the path to the BIS server program directory (the directory that contains `BISISAPI.DLL`). This field is preset to the directory where you last installed BIS. You can override this by pressing the **Browse** button and browsing to a new directory; by typing a directory name; or by typing the full path where `BISISAPI.DLL` can be found.

Creating the Directory

When all of the above fields are filled, click the **Create BIS Virtual Directory** button to begin the process of creating the virtual directory. Be patient-it can take 30 seconds to create the directory. Once the program finishes, messages will appear in the box at the bottom of the window. At that point, you can create another directory or close the program.

Testing the New Directory

To determine if the newly created directory is functional, create a text file named `default.srf` in the physical directory that you specified above. Type the following:

```
<html>
{{handler *}}
<head>
</head>
<body>
You requested page:
http://{{Value(HTTP_HOST,HTMLENCODE)}}{{Value(HTTP_URL,URLDECODE,HTMLENCODE)}}
</body>
</html>
```

Then enter the following into your web browser:

```
http://localhost/vdir
```

(replacing *`vdir`* with the name of your virtual directory).

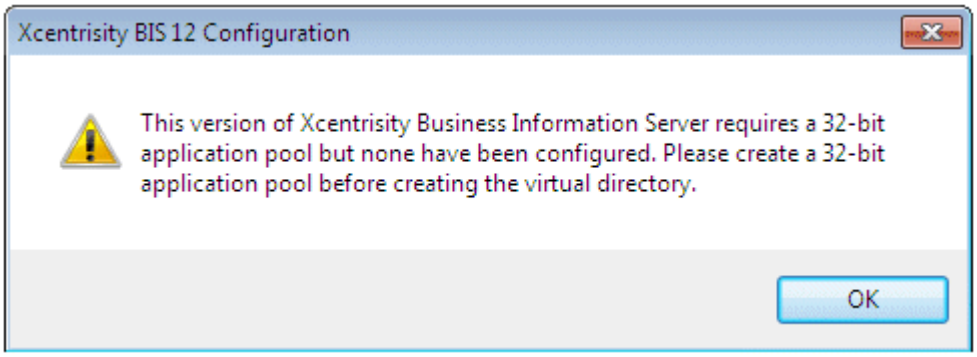
You should see a page containing only this text:

```
You requested page: http://localhost/vdir/
```

Notice how the `Value` tags were replaced with the server variables. If the `Value` tags were properly substituted, BIS is operational in this directory.

64-Bit Windows Considerations

On 64-bit versions of Windows that run Internet Information Server version 7 or later, BIS must be configured to run in a 32-bit application pool. If a 32-bit application pool is not detected when `BISMkDir` is launched, the following dialog will be displayed:



If you receive this warning, please use the IIS administrator to create a new application pool, and then use the **Advanced Settings** dialog for the new application pool and set **Enable 32-bit Applications** to **True**. Windows Security and Authentication

Building and Running BIS Samples

The BIS Samples include an installation verification application and several simple applications that illustrate the major Xcentrisity techniques for constructing web applications and services using BIS. These samples include complete source code as well as all of the XSLT transforms necessary to run them. In addition, each includes a batch file (or shell script) that will build the operational web application from source. This is convenient if you wish to experiment with modifications to the samples, or if you want to use the samples as the basis for your own web application.

If you choose to build a sample from source you must be sure that the environment variable `COBDIR` is set to the directory on your machine containing the Visual COBOL development system (with XML Extensions) that you wish to use. This environment variable may be set by the installation process, or it might have to be set manually prior to building the sample BIS application.

After verifying and setting `COBDIR` if necessary, ensure that a command prompt is present and the current directory is the `src` directory for the sample that you are building. Execute the script by typing:

```
build.bat
```

or (for BIS/Apache):

```
build.sh
```

After the processing has been completed and a command prompt appears, you will have rebuilt the sample and generated new files in the `bin` directory

Migrating from RM/COBOL to Visual COBOL

This section covers the process of migration of a BIS application from RM/COBOL to Visual COBOL. It does not cover the general COBOL differences between the two dialects, but just the BIS issues.

Note: The names of B$ functions are changed to B_ for Visual COBOL.

Using GOBACK

In Visual COBOL, a STOP RUN statement terminates the RUN UNIT unconditionally and will cause the service program not to return to the Service Engine properly. Service programs should exit using GO BACK instead.

Replace:

```
STOP RUN
```

With:

```
GOBACK
```

Compiling and Starting Service Programs

Under Visual COBOL, service programs must be DLLs. They must be compiled with the xmlgen(ws) Compiler option and then linked to produce a DLL. In the StartService tag, the `.dll` extension must be specified.

From a command file, the sequence is:

```
cobol.exe MyServiceProgram,MyServiceProgram.obj,,, xmlgen(ws) remove(name);
cbllink.exe -d -l MyServiceProgram.obj
```

In the SRF file, replace:

```
{{StartService(MyServiceProgram)}}
```

With:

```
{{StartService(MyServiceProgram.dll)}}
```

Passing Command Line Arguments

The parms section of the StartService tag is not supported by BIS for Visual COBOL. The recommended method of passing command line options is the ServiceOpts tag. However, this is not supported by BIS/IIS. For BIS/IIS, the recommended method of passing command line options, which Visual COBOL calls switches, is to use the SetEnv tag to set the COBSW environment variable.

The recommended method for passing a command line argument, the RM/COBOL A Option, is to set the value of the option in an environment variable, and then retrieve the value within the program using DISPLAY and ACCEPT. BIS/Apache also supports using the ServiceArgs tag.

In your SRF file, replace:

```
{{StartService(MyServiceProg A=ServiceArgument)}}
```

With:

```
{{SetEnv(MyArgument=ServiceArgument)}}
{{StartService(MyServiceProg.dll)}}
```

Then, in your service program, remove the LINKAGE SECTION and the USING from the PROCEDURE DIVISION.

Replace:

```
LINKAGE SECTION
01 A-PARAM.
   05 A-PARAM-SIZE         PIC S9(4) BINARY (2).
   05 A-PARAM-STRING.
      10 FILLER            PIC X OCCURS 0 TO 100 TIMES
                           DEPENDING ON A-PARAM-SIZE.
PROCEDURE DIVISION USING A-PARAM.
```

With:

```
01 A-PARAM                    PIC X(100).

PROCEDURE DIVISION.
BEGIN-PROG.
   DISPLAY "MyArgument" UPON ENVIRONMENT-NAME.
   ACCEPT A-PARAM FROM ENVIRONMENT-VALUE.
```

Retrieving the BIS_FILENAME

While it is possible to manage the Request and Response messages totally in memory using BIS for Visual COBOL, if it is desired to continue passing the messages in the file designated by the BIS_FILENAME environment variable, it will be necessary to change the code that retrieves the file name from the environment.

Replace:

```
CALL "C$GetEnv" USING "BIS_FILENAME",
                      BIS-Exchange-File-Name,
                      BIS-Exchange-File-Result.
```

With:

```
DISPLAY "BIS_FILENAME" UPON ENVIRONMENT-NAME.
ACCEPT BIS-Exchange-File-Name from ENVIRONMENT-VALUE.
```

Using In-Memory Messages

BIS for Visual COBOL allows Request and Response messages to be passed in memory, as passing messages containing sensitive information via a disk file is insecure. Also, although the latest version of BIS for RM/COBOL also supports in-memory messages, RM/COBOL's pointer support is different from Visual COBOL's, so further changes are required, namely the use of a second variable to pass the length of the message.

If it is desirable to change a BIS application to use in-memory message, the following changes are required.

1. Define pointer data items for pointers to the request and response messages.

   ```
   01 BIS-Response-String       POINTER.
   01 BIS-Request-String        POINTER.
   ```

2. Define numeric data items to contain the length of the request and response messages. Note that this step is also necessary when migrating from the BIS for RM/COBOL support for in-memory messages.

   ```
   01 BIS-Response-Len          PIC 9(5) Usage Comp-x.
   01 BIS-Request-Len           PIC 9(5) Usage Comp-x.
   ```

3. Locate all calls to `B_ReadRequest` and add the pointer and length parameters to them. (When migrating from BIS for RM/COBOL in-memory support, adding the length parameter is all that is necessary.)

   If the `B_ReadRequest` has a time-out parameter, replace:

   ```
   CALL "B_ReadRequest" USING Time-Out GIVING BIS-Status
   ```

   With:

   ```
   CALL "B_ReadRequest" USING Time-Out
                              BIS-Request-String
                              BIS-Request-Len
                        GIVING BIS-Status
   ```

   If the `B_ReadRequest` does not have a time-out parameter, replace:

   ```
   CALL "B_ReadRequest" GIVING BIS-Status
   ```

   With:

   ```
   CALL "B_ReadRequest" USING OMITTED
                              BIS-Request-String
                              BIS-Request-Len
                        GIVING BIS-Status
   ```

4. Locate the calls to XML IMPORT FILE and change them to be calls to XML IMPORT TEXT.

Replace:

```
XML IMPORT FILE
   SOAP-Request-Response
   BIS-Exchange-File-Name
   "SOAP-Request-Response"
   "soap_request_to_cobol.xsl".
```

With:

```
XML IMPORT TEXT
   SOAP-Request-Response
   BIS-Request-String
   BIS-Request-Len
   "SOAP-Request-Response"
   "soap_request_to_cobol.xsl"
```

5. Locate the calls to XML EXPORT FILE and change them to be calls to XML EXPORT TEXT.

   Replace:

```
XML EXPORT FILE
   SOAP-Request-Response
   BIS-Exchange-File-Name
   "SOAP-Request-Response"
   "cobol_to_soap_response.xsl"
```

   With:

```
XML EXPORT TEXT
   SOAP-Request-Response
   BIS-Response-String
   BIS-Response-Len
   "SOAP-Request-Response"
   "cobol_to_soap_response.xsl"
```

6. Locate the calls to `B_WriteResponse` and add the pointer and length parameters to it. (Again, for BIS for RM/COBOL in-memory support, only add the length.)

   If the `B_WriteResponse` has a disposition parameter, replace:

```
CALL "B_WriteResponse" USING Disposition GIVING BIS-Status.
```

   With:

```
CALL "B_WriteResponse" USING Disposition
                             BIS-Response-String
                             BIS-Response-Len
                       GIVING BIS-Status
```

   If the `B_WriteResponse` has does not have a disposition parameter, replace:

```
CALL "B_WriteResponse" GIVING BIS-Status.
```

   With:

```
CALL "B_WriteResponse" USING OMITTED
                             BIS-Response-String
                             BIS-Response-Len
                       GIVING BIS-Status
```

7. After each call to `B_WriteResponse`, free the memory associated with the response message by adding a call to XML FREE TEXT:

```
XML FREE TEXT BIS-Response-String
```

Replacing B_Exchange

`B_Exchange` has been deprecated. If it is present in the RM/COBOL program, it will need to be replaced with equivalent `B_` calls.

Replace:

```
Call "B_Exchange" using TimeoutInSeconds giving BIS-Status.
```

With:

```
call "B_WriteResponse" giving BIS-Status
if BIS-Status = BIS-Success or BIS-Status = BIS-Warn-ResponseUnexpected then
    call "B_ReadRequest" using TimeoutInSeconds giving BIS-Status
endif
```

Using B_Trace to Write to the BIS Trace

BIS for RM/COBOL could intercept DISPLAY messages and redirect them to the BIS trace file. This is not supported by BIS for Visual COBOL. Instead, replace the DISPLAYS with calls to `B_Trace`.

Replace:

```
Display "BIS Exchange File:" BIS-Exchange-File-Name
```

With:

```
CALL "B_Trace" Using "BIS Exchange File:" BIS-Exchange-File-Name
```

Glossary

Application Root Path
: A URL path that groups all of the pages of a BIS application. Under IIS, this is the URL path of the virtual directory that was specified during installation, or was created with the `BISMkDir` utility.

BIS Request Handler
: The BIS components activated when a Stencil (Server Response File) is the target of an HTTP request. The BIS Request Handler performs the processing of the Stencil, including the management of Sessions and the creation and destruction of Service Instances.

HTTP
: HyperText Transport Protocol, a standard protocol and encoding scheme used to transmit requests to web servers and receive responses from web servers. HTTPS is a secure version of HTTP.

Response Content
: The data included in the content area of an HTTP Response message.

Request Content
: The data included in the content area of an HTTP Request message.

Request Document
: An XML document produced by the BIS Web Server and including the information contained in an HTTP Request message as well as various values indicating the user agent and server environment in which the request was issues and is being processed.

Server Response File
: A file, usually with the extension `.srf`, which is used to direct the BIS Web Server in responding to a request. Also referred to as a Stencil.

Service Engine
: The BIS components responsible for performing the execution of a user-supplied Service Program and the synchronization and interaction between the Service Program and the BIS Web Server.

Service Instance
: An execution of a Service Program within a particular Session.

Service Program
: A user-supplied Visual COBOL program object file that is invoked by the BIS Request Handler and executed by the BIS Service Engine.

Session
: A *stateful* sequence of HTTP request/response interactions between a web user agent (for example, browser) and a BIS Request Handler. The session identification is preserved in the user agent by means of

a session cookie provided in the response to the first request of the session. All subsequent requests containing that cookie are assumed to be for the designated session.

Session Root Path

The URL path that contains the object that caused the current session to be created. For example, if the requested URL is http://localhost/xbisvcob/default.srf, the session root path is xbisvcob. By default, all pages that contain the session root path in their URL path will be served using the same session. This can be overridden by specifying `Scope=ISOLATE` in a `SessionParms` tag.

Stencil

A file, usually with the extension `.srf`, which is used to direct the BIS Request Handler in responding to a request. Also referred to as a Server Response File.

URI

A Uniform Resource Identifier, the naming convention for objects on the Internet. A URI consists of a *scheme*, followed by a colon, followed by a scheme specific name. A URI can be further classified as a Locator, or a Name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URI that identify resources via a representation of their primary access mechanism (e.g., their network "location"), rather than identifying the resource by name or by some other attribute(s) of that resource. The term "Uniform Resource Name" (URN) refers to the subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.

URL

A Uniform Resource Locator, the location of a resource on the internet. A URL is a type of URI (Uniform Resource Identifier), and consists of a *scheme* (in this context, HTTP or HTTPS), the name of a *machine* (sometimes also called the *authority*), and a path to a resource (for example, a file). For example, `http://localhost/xbisvcob/index.html` specifies the file named `index.html` from directory xbisvcob on server machine `localhost` using the HTTP scheme. When this is typed into a web browser, the browser issues an HTTP GET request on this resource.

URL Path

The path portion of a URL - that is, the part after the server identifier up to the end of the URL, the query string, or fragment (whichever comes first). For example, in the URL `http://localhost/xbisvcob/default.srf?query=yes#top`, the URL path is `xbisvcob/default.srf`.