Liant Software Corporation

Business Information Server User's Guide

Version 8

LIANT

Liant Software Corporation
8911 N. Capital of Texas Highway
Austin, TX 78759
U.S.A.

Phone	(512) 343-1010
	(800) 762-6265
Fax	(512) 343-9487
Web site	http://www.liant.com

Documentation Release History:

401223	Business Information Server User's Guide Version 8	February 2004
401230	Business Information Server User's Guide Version 8	May 2004

Table of Contents

Chapter 1. Introducing the Liant Business Information Server

1.1 Overview

The Liant Business Information Server (BIS) is a web server environment that manages COBOL application sessions and makes them available via any web browser that is granted access to the server. BIS offers application developers a real opportunity to build state-of-the-art browser-based applications or Simple Object Access Protocol (SOAP) implementations that include COBOL object programs and COBOL data.

With BIS, remote users can access data, perform application functions and execute COBOL programs on one or multiple servers located anywhere in the world. 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 will have access to public records from anywhere.

Liant BIS has two major components:

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

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 and sends the request to the server specified in the URL. If the requested page is a reference to a simple HTML file, the contents of the file are sent to the browser.

Some useful definitions:

User Agent / Client	The program that is used to request information from a server. This is frequently a web browser, but 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://liant.com/bis/index.html specifies the file named *index.html* from directory *bis* on server machine *liant.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, or may POST data (such as a form) to the server, or may cause a program to be run on the server. GET and POST are by far the most frequently used commands.
SOAP	SOAP 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.
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 (like 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"—the web server processes each request as if it had never received another 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.

See the **Glossary** on page 94 for more definitions.

1.2 Installation on Windows

This section details installation of Business Information Server on Windows. Linux installation is described in section 1.3.

1.2.1 Prerequisites

These are the prerequisites for BIS on Windows:

- A host machine running Windows 2000, Windows XP Professional, or Windows Server 2003 operating system. When BIS is installed on Windows 2000 Workstation or Windows XP, there are connection limit restrictions that prevent use as a real-world web server. These systems do work well for BIS application development and testing, however.

- Internet Information Server (IIS) must be installed. IIS is the Microsoft web server that listens for HTTP requests on port 80 and 443, and routes BIS requests to the BIS Web Server. BIS will not install unless IIS is already present. To install IIS, go to Start→ Control Panel→ Add or Remove Programs. Select the Add/Remove Windows Components button and follow the instructions to ensure Internet Information Server (IIS) is installed. A reboot will most likely be required.

- For Windows Server 2003 only, an additional step is required to allow BIS to run: ISAPI extensions must be enabled. These are enabled by default on Windows 2000 and Windows XP, but are disabled by default on Windows Server 2003. To enable these extensions:

  1. Select

     Start
     → Control Panel
     → Adminstrative Tools
     → Internet Information Services (IIS) Manager

  2. Expand **Local Computer**, and then click on **Web Service Extensions**.

3. In the window on the right, make sure the **Extended** tab at the bottom is selected. Then, click on **Add a new web service extension...** Type **srf** for the "Extension name" and type in the path to the BIS ISAPI plug-in DLL (usually **c:\Program Files\Liant\BIS\BISISAPI.dll**) in "Required files". Click the check box for "Set extension status to Allowed", and then click "OK".
4. Right-click on **Local Computer**, click on **All Tasks**, and select **Restart IIS**.
5. Close the Internet Information Server (IIS) Manager window. Configuration is complete.

1.2.2 Installation

The BIS installation consists of three components:

SETUPBIS.EXE	The installation program.
SETUPBIS.MSI	A Microsoft Installer package that contains the BIS programs and samples.
LIANT.LIC	The license file required by the BIS installation. The BIS installation will ask for this file unless it is located in the directory from which the BIS installer was launched.

To start the BIS installer:

- If you have CD-ROM media, insert the disk in the drive. If the BIS installer does not start after a few seconds, start it manually by using Windows Explorer to navigate to the CD drive. Then double-click on SETUPBIS.EXE.
- If you downloaded the installation program, use Windows Explorer to navigate to the directory that contains **SETUPBIS.EXE**. Then double-click on the program.
- At this point, you will see several setup windows, culminating in the dialog box in Figure 1-1.
- Note: In all BIS setup dialogs, press **Next** to move forward in the installation, and **Back** to revisit a previous step. Pressing **Cancel** at any point cancels the installation without making any changes to your system.

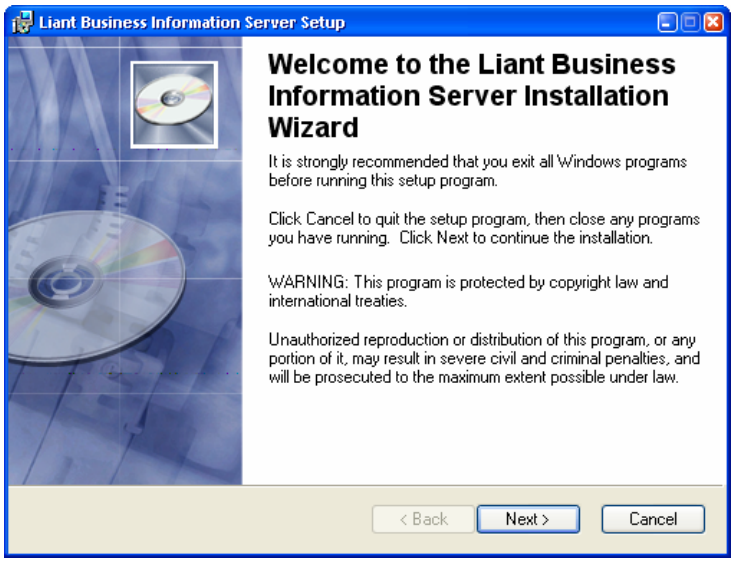


Figure 1-1. Installation Welcome Dialog Box.

- Press **Next**.

1.2.3 The License Agreement

The license agreement is displayed when you press **Next**. Please read it carefully, and if you agree, click the "I accept this license agreement" button and click **Next**.

1.2.4 READ ME Information

The next dialog contains important, late-breaking information about BIS. Please read it and press **Next**.

Note: if you would prefer to read this in a larger window, you can copy the text from the dialog box and paste it into *WordPad* or any word processor. To do this

1. Click in the README window.
2. Press **Ctrl+A** to select all text, and then press **Ctrl+C** to copy the text to the clipboard.
3. Start the *WordPad* program with **Start→ Run→ WordPad**.
4. You can now read or print the README documentation in WordPad.

When you are ready to proceed, press **Next**.

1.2.5 Liant License File

BIS installation requires a Liant license file, usually named **LIANT.LIC**.

At this point, enter the name of the license file. You can press the Browse button to search for it.

Note that the dialog at the right is not displayed if file **LIANT.LIC** is found in the directory from which the installer was launched.

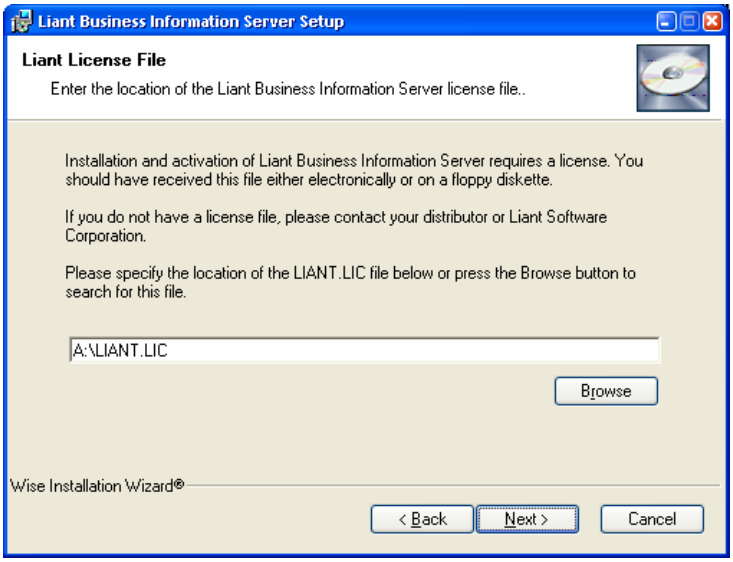


Figure 1-2. Installation License File Dialog Box.

1.2.6 User Information

Enter your name and the name of your organization and press **Next**.

1.2.7 Destination Folder

Choose the installation folder for the BIS program files. The default is

```
Program Files\Liant\BIS
```

We recommend the default be used. Press **Next** after making your selection.

1.2.8 Select Features

This dialog allows you to choose the features that will be installed on your server.

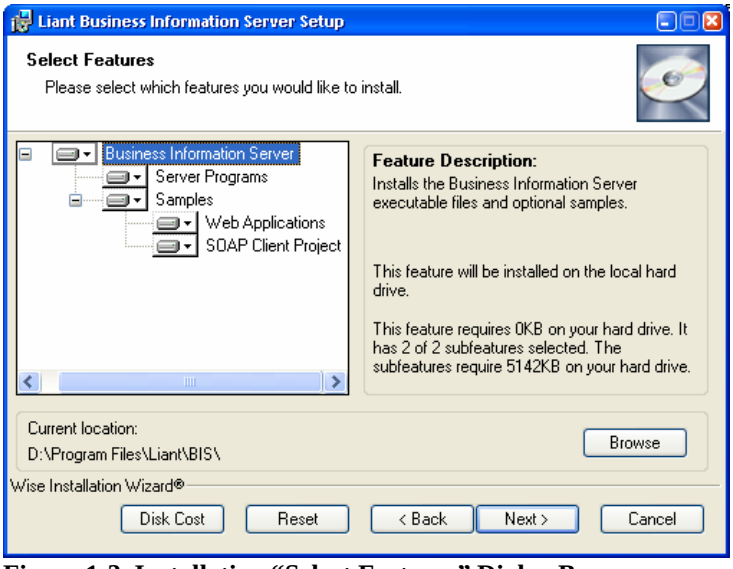


Figure 1-3. Installation "Select Features" Dialog Box.

- There are several features that may be installed:

1. **Server Programs** includes the BIS request handler and the service engine. This is a required feature and cannot be de-selected.

2. **Samples** is optional includes several sub-features:

    - **Web Applications** are installed in a newly created virtual IIS directory named **LiantBIS**. These sample web applications are installed by default because they can be used to quickly verify that the BIS request handler is operational. Note: please do not change the name of the directory or the samples will not be configured correctly.
    - **SOAP** Client Project is a sample .NET project. It builds a Calculator client program that calls a COBOL SOAP service in the SAMPLE3 Web Application installed above. Full source code is included and is installed by default in this directory:

```
My Documents\Visual Studio Projects\Liant\BIS\SoapSample3
```

    Note that *Microsoft Visual Studio.NET* (2003 or later) is a prerequisite for building this sample project If this feature selected, a pre-built calculator client is provided in the server programs directory and a shortcut is created under **Start→All Programs→Liant→BIS**.

Additional samples may also be available.

Note that you can also:

- Change the installation location for a feature or sub feature by pressing the *Browse* button.
- Press the **Disk Cost** button to see an overview of the amount of space available on your volumes.
- Once you have selected the features that you wish to install, press **Next**.

1.2.9 Logon Information

This dialog selects the Windows logon ID that will be used to run BIS services.

The account chosen must have sufficient privileges to access the .COB program files, and the data files that are required to service BIS requests.

In this dialog, you must:

- Enter the user name (logon ID) and password that the BIS service engine should impersonate when running programs. The installer will validate the username and password.

- To search for an existing user, press the **Browse** button. Enter the name of a domain, server, or press the browse button to select from a list. Then enter a user name or press the browse button to select from a list. Finally, press the **Ok** button to paste the result into the User Name field.

- To create a new user, press the **Create New User...** button. Select a domain or server, and specify a user name to create along with a password. Finally, select a group for the new user (or **None**).

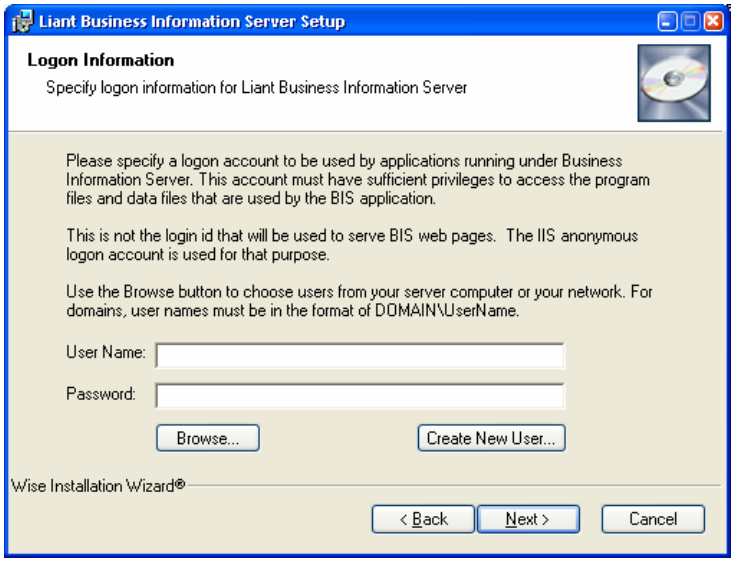


Figure 1-6. The Installation "Logon Information" Dialog Box.

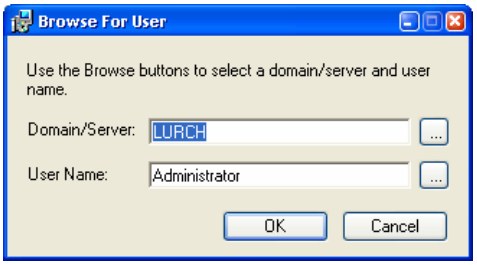


Figure 1-6. Installation "Browse for User" Dialog Box

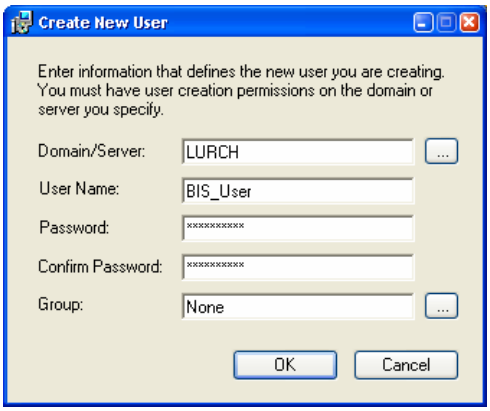


Figure 1-6. The Installation "Create New User" Dialog Box.

Once the **User Name** and **Password** have been selected, press **Next**. The installer will validate the information and report an error if the logon ID or the password is invalid.

Note: the logon ID can be changed at any time on the server—reinstallation is not required. See “Configuration after Installation (Windows)” on page 83 for more information.

1.2.10 Ready to Install

At this point, the BIS installer has all the information that is required to install BIS. If you are satisfied with the preceding choices, press **Next** to begin the installation.

1.2.11 Installation Complete

If you see a dialog stating “Liant Business Information Server has been successfully installed”, congratulations! You are ready to test the installation. If you **receive** another message, please see “BIS Troubleshooting Tips” on page 81 for assistance.

1.3 Installation on Linux

This section details installation of Business Information Server on Linux. Windows installation is described in section 1.2.

1.3.1 Prerequisites

BIS on Linux requires that the Apache web server be installed and that the mod_perl plug-in be statically linked with Apache. In addition, there are several mod_perl packages that must be installed before BIS can be successfully installed. More specifically, BIS for Linux requires:

- A host machine running the Red Hat Linux operating system versions 7.3 and above. BIS has been tested on Red Hat versions 7.3, 9, Enterprise and Fedora and should work on any Red Hat release with the large file kernel that can support the appropriate versions of Apache and mod_perl (See below).

- Perl 5.8.x must be installed. If it is not installed, go here[1] for more information.

- Libxml2 must be installed. If it is not installed, go here[2] for more information. Be sure that the installed version of libxml2 is compatible with XML::LibXML (the XML package for mod_perl). As of the 1.56 version of XML::LibXML, the following versions of libxml2 have been tested and are believed to work properly:

  2.4.20, or 2.4.21, or 2.4.23, or 2.4.24, or 2.4.26, or 2.4.27, or
  2.5.2 <= version < 2.5.5, or
  2.5.5 < version <= 2.5.10 where
  version = libxml2 version number.

  If an inadequate version of libxml2 is presently installed, you must completely obliterate it and install a compatible version before proceeding. If any remnants of the incompatible version are left around, XML::LibXML will probably not install correctly.

- The Apache 1.3.18+ web server must be installed. BIS has been tested on version 1.3.18, 1.3.28, 1.3.29. Note that BIS has not been tested on Apache 2 and should not be expected to work properly on that platform. Apache normally listens for HTTP requests on port 80 and 443, and when properly configured, routes BIS requests to the BIS Request Handler. BIS will not install unless Apache is already present. If your system does not have Apache installed, go here[3] for more information, but first, read on.

- The mod_perl 1.3.28+ Apache plug-in must be statically linked into your Apache. BIS has been tested on version 1.3.28 and 1.3.29. Note that BIS has not been tested on mod_perl2 and should not be expected to work properly. If you do not have Apache installed, or if the Apache installed does not have mod_perl linked in, the following steps will build a new Apache server with mod_perl suitable for BIS:

1.3.2 Installing an Apache/mod_perl Web Server for BIS

First, assuming you are in your home directory (**cd ~**) and have root privileges, do the following:

[1] http://www.cpan.org/src/stable.tar.gz
[2] http://xmlsoft.org/downloads.html
[3] http://httpd.apache.org/docs/install.html

```
mkdir bis-install
cd bis-install
perl –MCPAN –e shell
```

You'll be prompted for numerous answers; you should take the default for all of them except when it asks about your locale.

When it asks which servers you would like to use for retrieving Perl modules, just enter **1 2 3 4 5 6 7 8 9**.

If all this has completed successfully then you will have a prompt that looks like:

```
cpan>
```

Type **bye** at this prompt to leave cpan.

Now enter the following commands:

```
wget http://www.apache.org/dist/httpd/apache_1.3.29.tar.gz
wget http://perl.apache.org/dist/mod_perl-1.0-current.tar.gz
tar –xvzf apach*
tar –xvzf mod_*
perl –MCPAN –e "install CGI"
cd mod_*
perl Makefile.PL DO_HTTPD=1 USE_APACI=1 EVERYTHING=1
     APACHE_PREFIX=/usr/local/apache
make && make install
perl –MCPAN –e "install Apache::Test"
perl –MCPAN –e "install XML::LibXML"
```

At this point, your system is ready to install BIS.

1.3.3 Installing BIS

From the directory the BIS tar.gz was unpacked into, type:

```
sh install
```

Be sure to select the correct Apache binary such as /usr/local/apache (i.e., not one from a source directory). Use the defaults for everything. Especially, DO NOT CHANGE the /usr/local/liant/bis default directory.

1.3.4 Configuring Apache

You should edit the main **httpd.conf** configuration file to include the following lines:

```
<IfModule mod_perl.c>
  Include conf/bis.conf
```

```
</IfModule>
```

1.3.5 Starting Apache and BIS

Use the following commands to start the BIS server and Apache:

```
/etc/init.d/liantbis start
/usr/local/apache/bin/apache ctl start
```

Use the actual location of Apache if it is different from the above.

1.4 Testing the Installation

The samples are the best way to verify that BIS was successfully installed. There are two ways to launch the samples on the server:

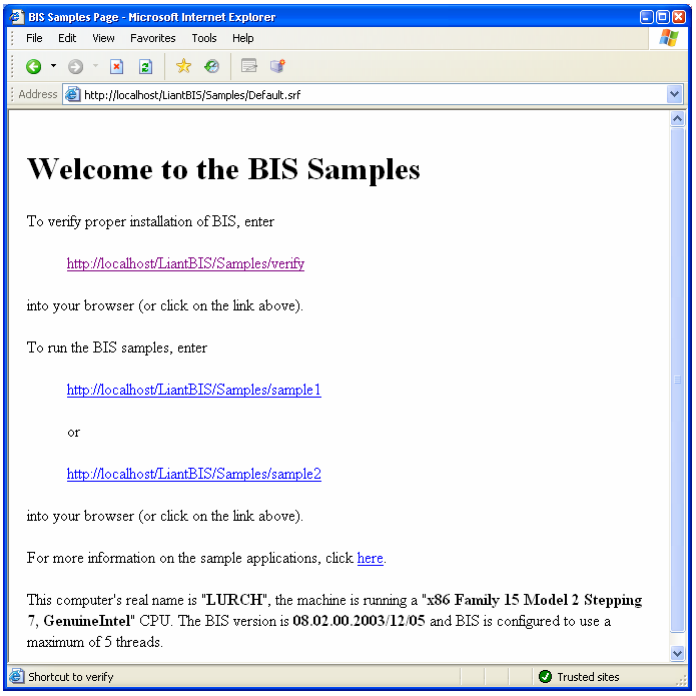


Figure 1-7. The BIS Samples Page.

- For BIS installed on a Windows system, click **Start→ All Programs→ Liant→ BIS→ BIS Samples**.
- For BIS installed on either Windows or Linux, start a web browser on the server and enter the following URL:

```
http://localhost/liantbis/samples/default.srf
```

  - A web browser should start and you should see the "Welcome to the BIS Samples" page illustrated in Figure 1-7.

1.5 Uninstalling BIS for Windows

To uninstall BIS for Windows, use the **Add or Remove Programs** control panel applet.

- Click **Start→Control Panel→Add or Remove Programs**
- Click on **Liant Business Information Server**
- Click the **Remove** button.

If you restart the **SETUPBIS.EXE** installation program and BIS is already installed, the installer will offer to **Modify**, **Repair**, or **Remove** the server. Selecting **Remove** is equivalent to removing BIS with the **Add or Remove Programs** control panel applet.

The web application samples are also removed, but the **SOAP Client Project** sample Visual Studio project file is not removed.

1.5.1 Removing only the Web Application Samples

To remove the samples from a Windows IIS website after installation, log onto the server. Then:

1. Click **Start → Control Panel → Administrative Tools → Internet Information Services**
2. Expand **Web Sites**, then **Default Web Site** (or your web site, if renamed).
3. Right-click on **LiantBIS** and select **Delete** from the popup menu.

Chapter 2. 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.

2.1.1 Web Protocols: Requests/Responses

Web clients and servers communicate by using a request/response protocol called **HTTP**, which is an abbreviation for **Hypertext Transfer Protocol**. HTTP includes two interfaces 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 interfaces (HEAD, PUT, DEBUG), but the above two are the ones used by BIS.

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**.
host	The name or location of the computer that will receive the request.
port	The TCP/IP port that the web server is “listening” to. By convention, this is port 80 if omitted. By requiring this parameter, a given host can support more than one web server.
absolute_path	The absolute location of the resource being requested on the host. This frequently, but not always is 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 type (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—then all of the above information (request type, URI, query parameters, form data) is passed to the BIS request handler, which then renders (i.e. executes) the control and substitution tags in that file. If BIS renders a **{{StartService}}** tag, a COBOL service program is started. If BIS subsequently renders an **{{XMLExchange}}**, the request is sent to the COBOL program, and the response is rendered into the page that is returned to the user agent.

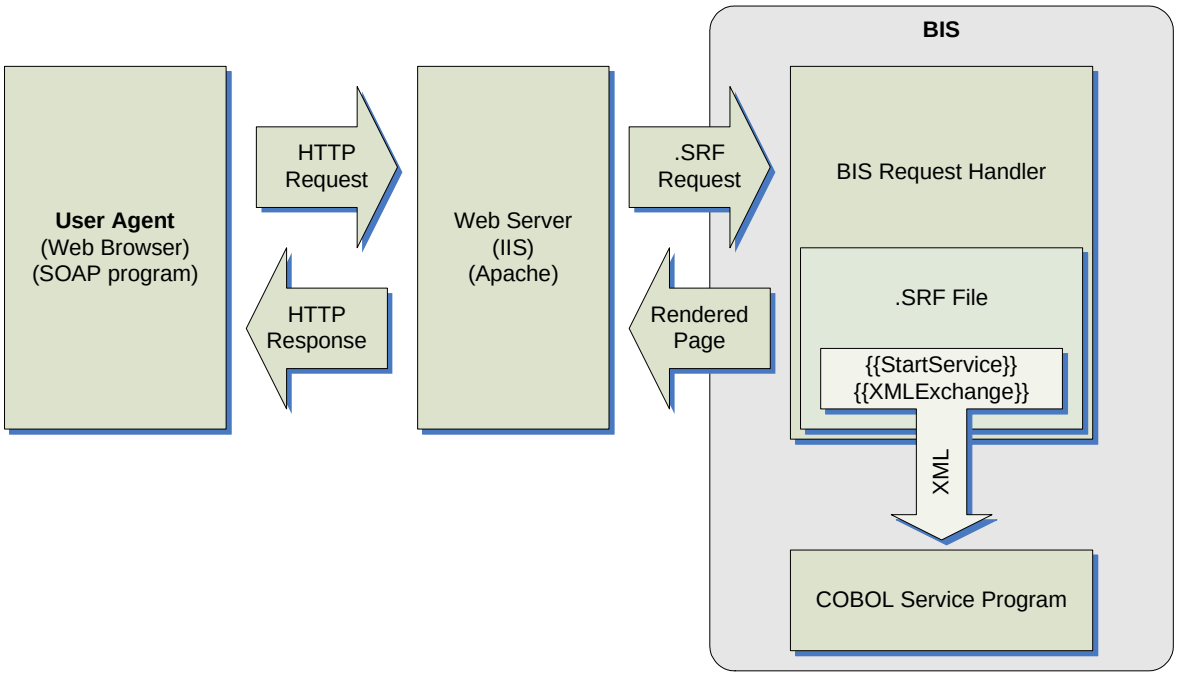


Figure 2-1. BIS Block Diagram.

2.1.2 Sessions

HTTP requests are innately stateless: the web server does not provide any built-in mechanism to group multiple requests together. However, once a service program is started, subsequent requests from the same user agent should probably be routed to the same service program. To make this possible, BIS creates an object on the server called a **Session** when a request first arrives from a particular user agent, and uses the Session to route subsequent requests from that user agent to the service program.

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 survives until

1. A predetermined but adjustable amount of time passes without an additional request from the user agent—referred to as the **Inactivity Timeout** period.

2. The session is explicitly destroyed: this can be requested by either the service program or by a special handler tag—the **{{SessionComplete}}** tag.

Open 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.

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. A cookie can be set in the user agent by the response. The user agent includes the cookie with the next request.

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

2.1.3 Cookies

When a client issues a request to the server, by default, BIS looks for a **Cookie** in the request to locate a session created by a previous request from the same user agent. A cookie is a specially named value that BIS includes with each response from the server to the user agent, and the UA will normally send the cookie in the next request to the same web server. When BIS 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 drawback of cookies is that some user agents purposely disable cookies for privacy reasons: unscrupulous websites 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 still conceivable that some user agents will not accept session cookies. This will unfortunately prevent BIS applications from working with that user agent.

2.1.4 Timeouts

BIS supports two kinds of timeouts

- Session Inactivity Timeouts
- Service Timeouts

These timeouts are described in detail in the following sections.

2.1.5 Session Inactivity Timeout

Session inactivity timeouts are used to detect abandoned sessions and free server resources by deleting those sessions. For example, each active [COBOL] 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.

Besides the inactivity timer, 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-ServiceComplete** 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.

2.1.6 Setting the Session Inactivity Time

The default inactivity timeout value for a BIS session is 600 seconds (10 minutes). However, this default 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. The value must be specified in seconds. For example:

  ```
  BIS_SESSION_INACTIVITY_TIMEOUT=600
  ```

  This environment variable sets the timeout to 600 seconds (10 minutes). See "Setting Environment Variables" on page 86 for information about setting and modifying environment variables on Windows and "Configuring Apache" on page 88 for information on configuring these variables on UNIX.

- The timeout may be set from within a **.srf** file (see Section 4) by using the **{{ SessionParms(InactivityTimeout=*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$SetInactivityTimeout(*seconds*)** call. Note that this call does not take effect until the next time the service program interacts with the BIS request handler—that is, the service calls **B$ReadRequest** or **B$Exchange** and the s renders an **{{XMLExchange}}** tag.

  Of these, the **BIS_SESSION_INACTIVITY_TIMEOUT** variable has the lowest priority and is overridden by either **{{SessionParms}}** or **the B$SetInactivityTimeout** call.

2.1.7 Service Timeouts

When the BIS request handler passes a request to a COBOL 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 increased or decreased 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. The value must be specified in seconds. For example:

```
BIS_SERVICE_TIMEOUT=30
```

This environment variable sets the timeout to 30 seconds. See “Setting Environment Variables” on page 86 for information about setting and modifying environment variables on Windows and “Configuring Apache” on page 88 for information on configuring these variables 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 [COBOL] 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 has the lowest priority and is overridden by either **{{SessionParms}}** or **the B$SetServiceTimeout** call.

Chapter 3. Server Response Files

3.1 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 referencing a Server Response File URI.

Server Response Files are often regular 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, RM/COBOL-based service programs whose execution is controlled and synchronized by BIS.

3.2 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 (depending on the tag type) parameters may be. 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:

```
{{ StartService (samp03, mylibrary.cob) }}
```

- 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 257 characters.

Important: both the opening **"{{"** and the closing **"}}"** tag delimiters must be contained on the same line—that is, a tag may not span lines. Use caution when creating tags with HTML editors that reformat HTML and make sure that the formatted did not split a tag across multiple lines. Some strategies to avoid line wrapping problems:

- Turn off line and word wrapping in your HTML editor for **.SRF** files.
- Don't use spaces in tags.
- Embed non-rendering tags (that is, tags that do not produce HTML output) in HTML comment sequences, as HTML editors will normally not reformat these. For example:

```
<!-- {{ StartService( MyVeryLongProgramName -c MyLongConfigFile.cfg ) }} -->
```

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

3.2.1 Rendering Tag Types

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 output unchanged.

3.3 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 processed as they are encountered.

The output placed on the page when a tag is rendered depends on the tag. **Processing Control Tags** are removed from the output entirely and are never sent to the user agent. **Substitution Tags** are replaced by content that is generated as each particular tag is rendered. This content may, of course, be empty.

Note that tags are order-dependent. A particular tag may affect how later tags are rendered: for example, **{{StartService}}** determines the service that **{{XMLExchange}}** uses. In addition, the **{{Handler}}** tag must be the first non-comment tag on every page, and must appear within the first 4096 characters of the page.

3.4 Processing Control Tags

Processing Control Tags control how the page is processed by the BIS Request Handler. There is a tag that determines 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 emitted response.

3.5 Substitution Tags

Substitution Tags are replaced with new text or HTML. These tags are replaced by output from the RM/COBOL service program or by the BIS request handler directly without program interaction.

3.6 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.

3.6.1 Pathnames

There are two kinds of pathnames used within tags:

- A fully qualified pathname begins with a slash, or a tilde (~) character. On Windows, 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 determined when the **{{StartService}}** tag is executed. If a **.srf** file is subsequently served from a different directory, the current directory of the Service Engine is not changed. However, any relative pathnames in the new **.srf** file are still interpreted relative to the directory that contains the **.srf** file.

On Windows, pathname resolution within the BIS service program is affected by the **{{RunPath}}** tag and the **EXPANDED-PATH-SEARCH**, **RESOLVE-LEADING-NAME**, and **RESOLVE-SUBSEQUENT-NAMES** configuration options. These may be used to great effect in conjunction with the **{{SetEnv}}** tag.

3.6.2 Referencing Files in System Locations

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

The following parameters may be used in pathnames on both Windows and UNIX.

Symbol	Description
~	If the pathname begins with this character followed by a slash, it is replaced by the home directory. This is the login directory of the current user. On UNIX, this is the same as the contents of the **HOME** environment variable. On Windows, it is the concatenation of the **HOMEDRIVE** and **HOMEPATH** environment variables. If none of these variables are set, the path is not altered.

Under Windows, the following environment variables are useful in pathnames. Note that **RESOLVE-LEADING-NAME** must be set in the service configuration file for these to be useful.

Variable	Description
ProgramFiles	The location of the Windows **Program Files** directory.
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 Windows, you can also define synonyms on the server using the **RMCONFIG** program, 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 **RESOLVE-LEADING-NAME=@** in your configuration file, then you can refer to the file "**abc.cob**" in this directory in a path by specifying "**@MyPrograms/abc.cob**". See "Setting Environment Variables" on page 86 for information about setting and modifying environment variables on Windows.

On UNIX, use the **bis.conf** configuration file to define BIS environment variables. See "Configuring Apache" on page 88 for details.

3.6.3 Predefined BIS Environment Variables

BIS adds the following to the environment on both Windows and UNIX. Note that these variables are dynamically set during execution and are only available in the service program. They will not be visible in your shell environment.

Variable	Description
BIS_PROGRAM_DIR	The directory from which the BIS Service Engine is loaded. On Windows, this will normally be the LiantBIS.exe program in "c:\program files\Liant\BIS".
BIS_FILENAME	The fully qualified name of the temporary file created for this session that will be used to exchange data between the BIS request handler and the COBOL service program. • When the RM/COBOL service program calls B$WriteResponse or B$Exchange, the BIS Web Server reads this file to obtain the content (XML, HTML or plain text) that will replace the {{XMLExchange}} tag in the .srf file. • When the B$Exchange call returns or the service program calls B$ReadRequest, the current web request document (XML) is written into this file. This includes any content such as the 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 directory. Both the BIS Web Server 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.

3.6.4 The RUNPATH

If a relative filename is specified, the BIS service attempts to locate the file by searching the directories specified by the RUNPATH environment variable. For full details of how the BIS service locates files, please see "Locating RM/COBOL Files" on page 2-9 of the RM/COBOL User's Guide. Note that the {{RunPath}} tag may be used to insert additional directories before the default RUNPATH from the environment.

3.6.5 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:

1. Is the tag 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.

2. Did the tag fail to perform the requested function?

   If a malformed tag is embedded in an HTML comment (see the example on page 22), 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.

3. 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 Windows, to direct trace output to a file, replace **page** with **file** (or specify both with **page,file**). This will direct all trace output for the session into a file in the server's temporary directory (normally **C:\Documents and Settings*logonID*\Local Settings\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.

- 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 redirected with the trace **dir=** parameter. So on UNIX, **{{Trace(start)}}** is sufficient to create a trace file.

  Note that on UNIX, the **MASTER_DEBUG** configuration option must be enabled before any tracing can occur. See Appendix H.1, "Configuring Apache" in for details about setting or clearing this option.

Of all these techniques, tracing is the most useful and should be enabled during the development process.

Chapter 4. Tag Reference

This section presents and discusses each tag that is implemented by Business Information Server.

Here is an example of a basic **.srf** file. Tags are illustrated in red.

```
<!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 * }} -->
<!-- {{ StartService(samp03 -v,xmlif) }} -->
<!-- {{ Trace(queryparam=trace) }} -->
<html>
<head>
    <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Liant RM/COBOL Web Server Demonstration Page</title>
</head>

<body>
    <div align="center">
        <h3>Liant RM/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®* or *Macromedia Dreamweaver®* from reformatting the text in the handler tag, or possibly splitting the tag across multiple lines. Future releases of these products may support tags directly.

4.1 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 tells the web server that this particular **.srf** file will be served by Liant BIS.

```
handler *
```

In this release of BIS, the only supported handler tag is **{{ Handler * }}.** Future versions of BIS may support additional handlers.

4.1.1 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.

- Only one handler tag in each **.srf** file is permitted.

4.2 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.

4.2.1 Examples

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

4.2.2 Notes

- If not specified, the default content type is "**text/html; charset=utf-8**".
- If **{{ContentType}}** is specified multiple times on a page, the last instance is used.

4.3 The {{SessionParms}} tag

This tag allows various session attributes to be set.

```
sessionparms ( InactivityTimeout=seconds|DEFAULT,
               ServiceTimeout=seconds|DEFAULT,)
```

Where:

InactivityTimeout	Determines how long a session survives without user interaction. The default setting is 10 minutes.	
	DEFAULT	Resets the timeout to the default setting for the web.
	seconds	is an integer that specifies the number of seconds before the session terminates.
ServiceTimeout	Determines how long the service engine is given to perform its processing when a request is received.	
	DEFAULT	Resets the timeout to the default setting for the web.
	seconds	is an integer that specifies the number of seconds before service engine termination processing begins.

4.3.1 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.

4.4 The {{StartService}} tag

Starts the execution of a service program. Options and the names of one or more libraries to be used by the service can also be specified.

```
StartService ( program [parms] [,library1 [,library2]…] )
```

where:

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 parameters. Any text starting at the first non-blank after ***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. Some useful options:	
	-v	Causes additional trace information to be emitted if tracing is enabled. The additional information includes the names of autoloaded DLLs.
	-k	Suppresses the banner in trace information.
	-c **-x**	Specifies configuration files. Note that if a configuration filename is specified without path information, BIS will search the **RUNPATH** for this file.
library	A comma-separated list of service libraries. Do not include **–L**. If no extension is specified, under Windows, BIS will append **.dll** and UNIX will append **.so**. Note that this command line portability is an advantage in enumerating the libraries separately instead of with –l.	

BIS only allows one service to be active in a session.

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

When a service is started, BIS saves the name of the program. When another service 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 is started, page rendering resumes. Rendering and the service program run in parallel.

Examples:

```
1. {{ StartService (myapp) }}
2. {{ StartService (myapp, mylibrary.cob) }}
3. {{ StartService (myapp.cob, xmlif.dll, mylibrary.cob) }}
4. {{ StartService (myapp –V –C rmtcp32.cfg, xmlif.dll) }}
```

In these examples, the **.COB**, **.DLL**, and **.CFG** files must be in the **RUNPATH**.

1. Starts the program in file **myapp.cob**.
2. Attempts to start program **myapp** after loading the **mylibrary.cob** service library. If the library contains a program called **myapp**, it is run from the library. If the program is not in the library, then the first program in **myapp.cob** is started.
3. Starts the program in **myapp.cob** after loading **xmlif.dll** and **mylibrary.cob.**
4. Starts program **myapp.cob** after loading **xmlif.dll**. The **–V** option causes extra information about loaded programs to be emitted after the banner is emitted into the trace file. The **rmtcp32.cfg** file is processed when the service engine is loaded.

The **StartService** tag follows all the regular service engine program loading rules, including the **RMAUTOLD** directory. See the RM/COBOL Users' Guide for a detailed description.

Note that example 4 demonstrates a program that uses InfoExpress®. The **rmtcp32.cfg** file (which can have any name) should contain a line like this:

```
EXTERNAL-ACCESS-METHOD NAME=rmtcp32
```

On Windows, this causes **rmtcp32.dll** to be loaded when the service engine is loaded. This DLL implements the InfoExpress file access method.

4.4.1 Accessing the REQUEST from the Service Program

In many cases, the service program will require 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$ReadRequest** or **B$Exchange** within the service program. ...

4.4.2 Notes

- A given server response file can have at most one **{{StartService}}** tag. Additional **{{StartService}}** tags are ignored if a service is already running.

- The **{{StartService}}** tag must precede any tags that depend on the service program being active. Such tags currently include **{{XMLExchange}}**.

4.5 The {{RunPath}} Tag

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]… ] ) }}
```

4.5.1 Notes

- This tag causes the specified list of directories to be prefixed before the contents of any existing **RUNPATH** environment variable that is inherited from the system environment. Any number of directories may be specified, separated by commas or semi-colons (however, note that colons are not separators). If any *dir* contains spaces characters, it must be surrounded by double quotes. Directory names may not contain commas or semicolons.

- This tag is a session attribute and remains in effect until the session ends or another **RUNPATH** tag is encountered. To clear the run path, specify **{{ RunPath() }}**. Note that the **.srf** directory cannot be removed from the **RUNPATH** sent to the service program.

- This tag must precede the **StartService** tag or it will be ignored by the application.

- Relative directories in the list are interpreted to be relative to the directory that contains the **.srf** file for the page being processed. This is the current directory that is set when the service engine begins to execute.

- To explicitly reference the directory that contains the current **.srf** file, add "." (i.e. "current directory") to the path.

- See "Setting Environment Variables" on page 86 for information about setting and modifying environment variables on Windows.

4.6 The {{SetEnv}} Tag

This tag is used to set a variable in the service program's environment. Environment variables are treated as **synonyms** by the service engine.

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

4.6.1 Examples

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

4.6.2 Notes

- The **{{SetEnv}}** tag affects only the service engine's environment and not the BIS 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.

- The scope of a **{{SetEnv}}** tag is the current request, not 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.

4.7 The {{XMLExchange}} Tag

This tag causes the web server to request an XHTML form or table from the currently running RM/COBOL program. The XHTML form or table generated by the COBOL program replaces the **XMLExchange** tag in the output stream.

```
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 against the specified URL.

4.7.1 Notes

- Do not use **OnExit** with SOAP requests. SOAP clients may not be able to interpret the 302 error that is returned.

- The **OnExit** in the first **{{XMLExchange}}** following a **{{StartService}}** is ignored. This allows any service startup errors to be reported and corrected.

4.7.2 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 (and display 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.

4.8 The {{StopService}} tag

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

```
{{ StopService }}
```

4.8.1 Notes

- If the service program is not awaiting a request when this tag is rendered, the program must call **B$ReadRequest** or **B$Exchange** 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}}** immediately after a **{{StopService}}** 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 RM/COBOL program attached to this session.

4.9 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 }}
```

4.9.1 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.

4.10 The {{Trace}} Tag

Enables or disables trace logging for the current session.

```
{{ trace( options ) }}
```

These options control the internal accumulation of trace information on UNIX. Windows always accumulates trace information and these options are ignored.

START **STOP**	Starts/stops the accumulation of **TRACE** information. • **START** causes BIS to begin accumulating trace output. • **STOP** causes BIS to stop accumulating trace output. If tracing has been started, **START** has no effect. If tracing has not been started, **STOP** has no effect.
OFF	Turns tracing off. Equivalent to **STOP,NOPAGE,NOFILE,NOTAG,NOEXCHFILES,NOQUERYPARAM,NOIP**.

These options determine where the **TRACE** output is emitted.

NOPAGE **PAGE** **NOFILE** **FILE** **NOTAG** **TAG** **NOEXCHFILES** **EXCHFILES**	Controls **TRACE** output. • **PAGE** indicates that the trace is emitted at the end of the page. **NOPAGE** (the default) disables end of page trace output. • **FILE** indicates that the trace is written to a file in directory **DIR** (see below). **NOFILE** (the default) disables trace output to the file. • **TAG** enables the **{{TraceDump}}** tag and therefore determines if trace output is written when this tag is rendered. **NOTAG** (the default) causes **{{TraceDump}}** tags to be ignored. • **EXCHFILES** determines if a copy of the XMLExchange request/response files for each session are copied into the trace directory on Windows or emitted into the trace file on UNIX. **NOEXCHFILES** (the default) disables the tracing of XMLExchange request/response files.

These options are independent.

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

NODIR|DIR=*dir*	***dir*** specifies the directory that will receive trace output if **FILE** is in effect. If omitted, or if no ***dir*** is specified, but file output is enabled with either **FILE** or **EXCHFILES** then all trace output is written into the Windows temporary directory. If a relative directory is specified, output is written into a directory relative to the Windows temporary directory. If an absolute path is specified, output is written into that directory. In all cases, the specified directory is created if it does not exist.

These options allow tracing to be controlled using a query parameter:

NOQUERYPARAM **NOQP** **QUERYPARAM=*value*** **QP=*value*** **NOIP** **IP=xx.xx.xx.xx [-x.xx.xx.xx],...**	These options allow tracing to be dynamically enabled and disabled in a query parameter. The following options can be set in the query parameter: **START**, **STOP**, **RESET**, **PAGE**, **FILE**, **DUMP**, and **EXCHFILES.** • **QUERYPARAM** and **QP** are synonymous and select a query parameter that can be used to dynamically specify the options above. To disable the query parameter, use **NOQUERYPARAM** or **NOQP**. Trace options set in the URL 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. • **IP=xx.xx.xx.xx** 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.

4.10.1 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 (i.e. **PAGE**, **FILE**, **TAG, EXCHFILES**) are specified.

- The trace mode is part of the session and is "sticky" – that is, 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 a **.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.

4.10.2 Examples

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

4.10.3 The {{Trace}} Query Parameter

If the query parameter has been enabled for this session, the presence of the query parameter on a subsequent URL acts to change the trace options at the time the request is processed.

4.10.4 The BIS_TRACE_SUFFIX environment variable

The **BIS_TRACE_SUFFIX** environment variable 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” on page 86 for information about setting and modifying environment variables on Windows.

4.11 The {{TraceDump}} Tag

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

```
{{ tracedump }}
```

4.11.1 Notes

- This tag is ignored (i.e. 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 **{{TraceDump}}** to be specified near the end of the page.
- If this tag is omitted and tracing is enabled, BIS for Windows appends trace output to the end of the response (that is, after the **</html>** tag).
- **{{DumpTrace}}** is an alias for **{{TraceDump}}**.

Chapter 5. Conditional Tags and Constructs

Conditional tags evaluate a condition and return either **true** or **false**. They are used in the following constructs.

5.1 The {{If}} / {{Else}} / {{EndIf}} tags

These tags can be used to conditionally show or hide sections of the .srf file.

```
{{ if tag }}
     if-content
{{ else }}
     else-content
{{ endif }}
```

5.1.1 Notes

- The **{{Value}}** tag is the only replacement tag currently supported in the **{{if}}** tag.
- The definition of *content* includes both HTML and replacement tags.
- Any HTML code in a skipped section is ignored and is not transmitted to the user agent. Rendering 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. These blocks can also be nested.
- 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 }}**

5.2 The {{While}} / {{EndWhile}} tags

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

```
{{ while tag }}
     content
{{ endwhile }}
```

5.2.1 Notes

- The **{{Value}}** tag is the only replacement tag currently supported in the **{{while}}** tag.
- The definition of *content* includes both HTML 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.

Chapter 6. Substitution Tags

Substitution tags are replaced in the output stream. These tags can appear anywhere in the **.srf** file – even before the **handler** tag.

6.1 The {{Value}} tag

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

```
{{ VALUE (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, or the server configuration.

The **source** option determines from where the **variable** is obtained.

SERVER	Specifies that **variable** is a server variable. This is the default if none of the other sources below are specified. Under Apache, **ENV** and **SERVER** are identical.
CONFIG	Specifies that **variable** is a special server configuration value. A list of these 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 Apache servers, **ENV** and **SERVER** are identical. See “Setting Environment Variables” on page 86 for information about setting and modifying environment variables on Windows.
FORM	Specifies that **variable** is a <form> variable.
QUERYPARAM	Specifies that **variable** is a URL query parameter.

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 if the variable is not defined. • If **DEFAULT** is not specified and no regular expression is specified, the tag is simply removed from the output. • If **DEFAULT** is specified, the tag is replaced by ***value***. Note that, since ***value*** must be specified as a constant, additional operations (i.e. **GETDIR**, **TOUPPER, URLENCODE**) are not performed on **value**.
GETDIR	If **variable** is a pathname, this option extracts the directory portion of the pathname using the following algorithm:

	• Any query parameter is removed. This consists of truncating the pathname at the left-most question mark (?) character. • The right-most forward or backwards slash is located. If found, the pathname is truncated at that point (note the slash is removed as well). Note that, since **GETDIR** assumes that the variable may be a URL, it will not work as expected on pathnames that have embedded question marks. The results are undefined if the variable does not contain a pathname.
GETNAME	If **variable** is a pathname, this option extracts the filename portion of the pathname. The processing is the same as **GETDIR** except all characters after the directory but preceding the query parameter are returned. The results are undefined if the variable does not contain a pathname.
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: see 72
TOUPPER	Converts the value to all upper-case characters. Equivalent to **CHANGE="/.*/\U&/"**.
TOLOWER	Converts the value to all lower-case characters. Equivalent to **CHANGE="/.*/\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 &ltTest String&gt."**

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 Appendix D.

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).

On WindowsXP,

```
{{ VALUE (PROCESSOR_IDENTIFIER, ENV, DEFAULT="Unknown", HTMLENCODE) }}
```

is replaced by the contents of the **PROCESSOR_IDENTIFIER** environment variable. If this variable is not defined, the text **Unknown** (without quotes) is output instead. The output is HTML-encoded so any '<' or '>' characters in the environment variable are properly converted.

6.1.1 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 **FIND** 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:

  ```
  {{ VALUE (QUERY_STRING, FIND="?userid=fred\s", URLENCODE) }}
  {{ VALUE (QUERY_STRING, FIND=/?userid="fred\s/, URLENCODE) }}
  ```

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

6.1.2 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. This server variable may be undefined under Apache.
MAXTHREADS	[IIS Only] 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).
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** keyword.
SERVICENAME	Retrieves the name of the currently active service. If there is no active service program, the value is considered undefined and may be overridden with the **DEFAULT** keyword.
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.

6.2 The {{ Include }} tag

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

```
{{ include filepath }}
```

Where ***filepath*** is the path to the file whose content will be included in the response at the position of the **include** tag. You may specify an absolute path or a path relative to the physical location of the **.srf** file.

If the included file is a **.srf** file, tags in that file will also be expanded. However, the scope of a handler tag only encompasses the **.srf** file that contains the handler tag. This means that any included **.srf** file must also contain a handler tag (specified relative to the location of that **.srf** file) in order for its tags to be recognized and processed. Normally, specifying **{{Handler *}}** in each **.srf** file is sufficient.

6.2.1 Notes

- If ***filepath*** specifies a server response file, tags in that **.srf** file are also processed.
- Relative pathnames in ***filepath*** are interpreted as relative to the location of the top-level **.srf** file.
- An **include** tag can appear anywhere in the **.srf** file—even before the **handler** tag.

6.3 Comment tags

This tag is ignored and is simply removed from the output. This differs 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 }}
```

6.3.1 Notes

- A comment tag can appear anywhere in the .srf file—even before the **handler** tag.

- If a comment tag is immediately followed by the end-of-line character, BIS removes the end-of-line character along with the comment tag from the output. This is useful when placing tags into a file where white space is significant. For example, the **default.srf** file in **SAMPLE2** is 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) }}{{//}}
{{ RunPath(bin,../common) }}{{//}}
{{ StartService(webappsample2 -v) }}{{//}}
{{ XMLExchange(OnExit="gotit.srf") }}
```

Here, the comment tags and the **Handler**, **Trace**, **RunPath**, and **StartService** tags are completely removed from the output, while the **XMLExchange** 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 six blank lines before the start of the XML produced by the **XMLExchange** tag.

To avoid this in this sample, the non-comment **Handler**, **RunPath**, and **StartService** tags are followed by empty comments, which suppress the new characters. The XMLExchange 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.

Chapter 7. Service Programs

7.1 Introduction

The **Service Engine** is the BIS component that starts and runs service programs in response to requests. Currently, all BIS service programs are RM/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 either **B$ReadRequest** or **B$Exchange**.

The simplified flow of control is depicted below.

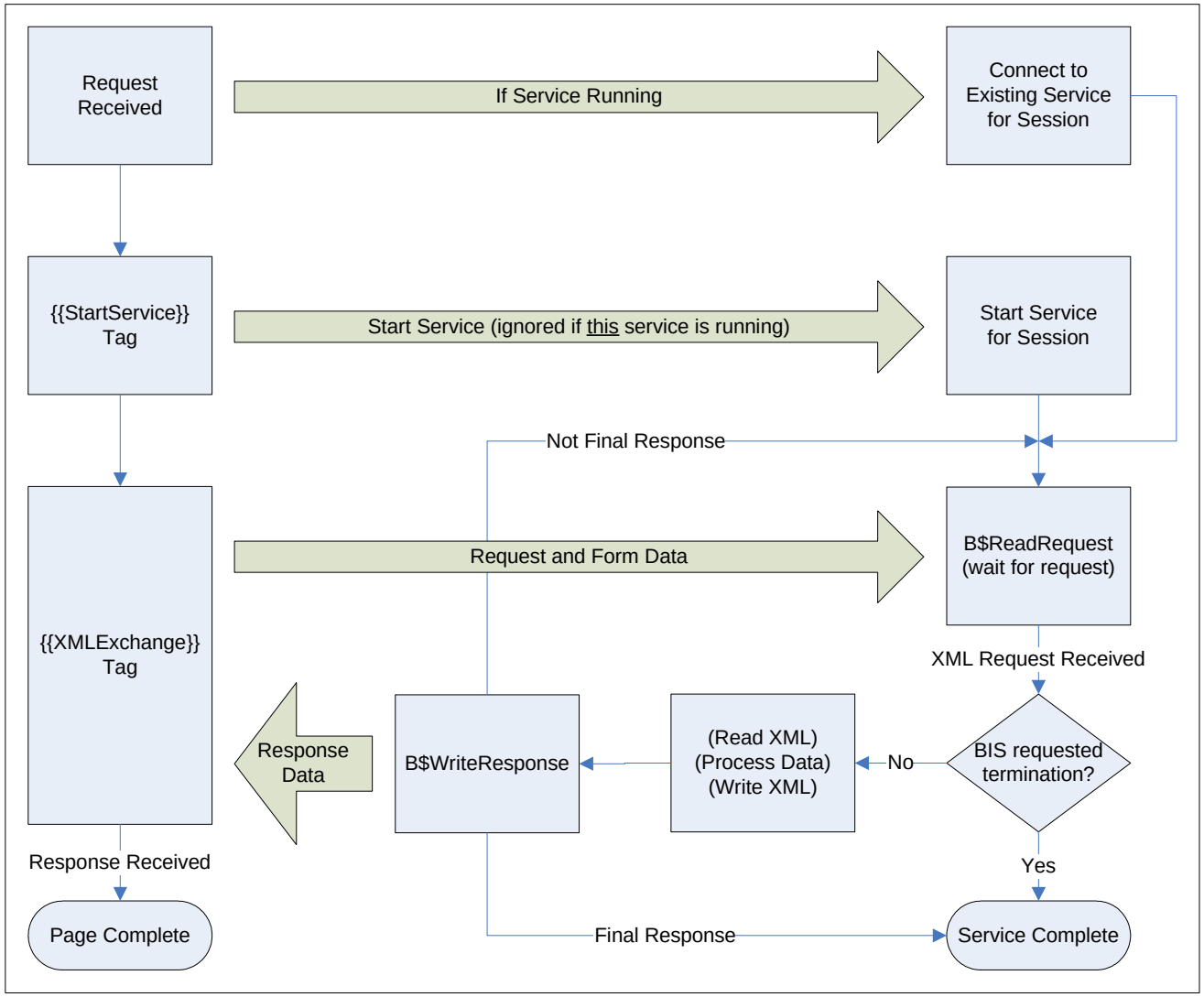


Figure 7-1

The BIS request handler and the BIS service engine synchronize when the request handler renders an **{{XMLExchange}}** tag and the service engine calls either **B$ReadRequest** or **B$Exchange**. Ideally, the

service engine will be waiting at a synchronization point when BIS is ready to provide a request. To avoid deadlocks, once BIS begins to process the **{{XMLExchange}}** tag:

- The service program must call one of the above **B$** functions within *ServiceTimeout* seconds.
- Alternatively, the program may request additional time by calling "**B$SetServiceTimeout" using 0** to reset the timer.

Once the service engine has read the request, it is granted a new *ServiceTimeout* interval to read the XML request, compute the response, write the XML response, and call **B$WriteResponse** or **B$Exchange**. Alternatively, the service program can terminate, which will cause the BIS service 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 BIS 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$ReadRequest** or **B$Exchange**.
- Terminate (for example, with **STOP RUN**.)[4]

If neither of the above two events occurs, BIS will terminate the service engine.

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

[4] Termination between requests in this way is not recommended because this is not deterministic—ideally, the program will terminate before BIS renders another **{{XMLExchange}}**, but this cannot be guaranteed. It is much better to inform the BIS service handler that the program is complete by setting the *ProgramDisposition* or *SessionDisposition* parameter in the final call to **B$WriteResponse**.

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 *ServiceTimeout* seconds to clean up and perform a STOP RUN. If the program exceeds the allotted time, BIS requests that the program stop at the next statement boundary and the service timer is started again[5]. 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.

7.2.1 ACCEPT and DISPLAY Statements

DISPLAY statements are allowed in service programs and the data that would normally be displayed on the console is instead placed into the BIS trace output. This is a useful way to debug the service program but this technique cannot be used to communicate with end users.

Because the service program does not have access to the console or the Windows desktop, **ACCEPT** statements are ignored and are treated as if the console operator pressed the Return or Enter key without actually entering any data. Otherwise, the service program would stop on an ACCEPT, waiting for a response that cannot come.

Note that it is still possible for a service program that uses **ACCEPT** statements to hang, if the program loops back to repeat the **ACCEPT** upon receiving a zero-length response. For this reason, it is best to add code to skip around **ACCEPT** statements if the program is running under BIS.[6]

7.2.2 Windows Message Boxes and Dialogs

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. To avoid this problem, BIS detects that the service program is attempting to create a dialog or message box and denies the request.

7.3 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. BIS takes the current request, encodes it using XML, and places the request into this file when the service program calls **B$ReadRequest** or B**$Exchange**.

Important: the file is not created until one of the above two functions is called.

BIS places the fully qualified name of this file into the **BIS_FILENAME** environment variable when the service engine is started. It is therefore accessible to the RM/COBOL program via the **C$GetEnv** function:

[5] Note that, in this mode, the service engine will close any open files.
[6] Such a program will be automatically terminated by BIS if it does not call **B$ReadRequest** or **B$Exchange** within *ServiceTimeout* seconds after being started.

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

The value of this variable is the fully qualified pathname of the file and, on Windows, 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.

Some notes:

- You do not provide this environment variable. BIS will automatically create this file and set the environment variable when a service program is started.
- 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.
  - **B$Exchange** to both receive requests and write responses.
- The file is not created until **B$ReadRequest** or **B$Exchange** is called for the first time by the service program.

7.4 BIS Return Codes

Here are the return codes for the **B$** functions. These codes are defined in the **BISDEF.CPY** COPY file.

00-09	Success! For **B$ReadRequest** and **B$Exchange**, the request data is available in the XML exchange file. For **B$WriteResponse**, the response was accepted by the server.	
	00	**BIS-Success**
		The data transfer succeeded and the XML file contains the result of the operation.
10-19	A non-fatal event occurred, and recovery is possible.	
20-29	A failure occurred and the program may be able to recover. The XML file was not updated. These are less serious than later return codes.	
	20	**BIS-Warn-RequestTimeExpired**
		A timeout parameter was specified on the **B$ReadRequest** or **B$Exchange** 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$ReadRequest** and the service program has not responded.
	22	**BIS-Warn-ResponseUnexpected**
		The service program called **B$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 and the program may or may not be able to recover. The XML file was not updated.	
	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**
		BIS could not write the XML Exchange file.
	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 open or write the trace file.
	49	**BIS-Fail-FileFormat**
		The XML Exchange file format is invalid.
50-79	A failure occurred and the program cannot continue. The XML file was not updated and the program should terminate.	
	50	**BIS-Fail-SessionAbandoned**
		The session timed out.
	51	**BIS-Fail-SessionComplate**
		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—an **{{StopService}}** tag may have been processed.
80-99	A serious error occurred. The XML file was not updated and the program must terminate.	
	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 server.
	88	**BIS-Fail-ServerInternalError**
		An internal error occurred while the service program was communicating with the BIS server.
	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 server.
	91	**BIS-Fail-ServerBadLength**
		An internal error occurred while the service program was communicating with the BIS server.
	92	**BIS-Fail-ServerBadParameter**
		An internal error occurred while the service program was communicating with the BIS server.
	93	**BIS-Fail-ServerWrongMsg**
		An internal error occurred while the service program was communicating with the BIS server.
	99	**BIS-Fail-ServerConnectionLost**
		The connection between the BIS service program and the BIS server failed.

7.5 Service Program Functions

The following COBOL-callable functions may be used in BIS service programs to communicate with BIS.

- **B$ReadRequest**

- **B$WriteResponse**
- **B$Exchange**
- **B$SetInactivityTimeout**
- **B$SetServiceTimeout**

These functions are detailed in the following sections.

7.6 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] giving BIS-Status.
```

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

Event	Action
BIS 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.
BIS 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 *ServiceTimeout* seconds to do so.

The most common result codes (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.
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, but that normally indicates a serious problem has occurred.

When execution resumes and the result code is BIS-Success, the file specified by the BIS_FILENAME environment variable contains the request in XML format. The exact format of the request is described in "Format of a Request" later in this section.

7.6.1 Notes

- BIS 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$WriteResponse
  Call B$Exchange (a shorthand way of calling B$WriteResponse followed by a call to B$ReadRequest)
  Call B$SetServiceTimeout. In particular, a parameter of 0 will reset the timer, and start another *ServiceTimeout* interval.
  Terminate the program.

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

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

  Note that specifying a timeout 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.

7.7 B$WriteResponse

This function call transmits a response to BIS to be inserted into the output stream, replacing the **{{XMLExchange}}** tag in the output stream. The response must be written into the request file (specified by the **BIS_FILENAME** environment variable) before **B$WriteResponse** is called.

The response file will typically contain an HTML or XHTML block to be inserted into the **.srf** file that was requested but it may also contain a SOAP result or anything else that is meaningful to the HTML client program that issued the request.

The syntax of this function call is:

```
Call "B$WriteResponse" [using ProgramDisposition] giving BIS-Status.
```

If this is the final call to **B$WriteResponse** by this service, the optional **ProgramDisposition** parameter should be used to inform BIS that the service program is finished and optionally if the session should be destroyed. 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 most common result codes (see BIS Return Codes for a complete table):

BIS-Status Code	Event Description
BIS-Response-Normal	The default: BIS makes no assumptions about what the service program will do next. However, the service program is granted only *ServiceTimeout* seconds to exit or to read the next request.
BIS-Response-ServiceComplete	BIS assumes that the service is about to terminate. If the **{{XMLExchange}}** tag contains an **OnExit parameter**, the **OnExit** is taken immediately and any XML output is ignored. If there is no **OnExit**, the XML output is written to the output stream, completely replacing the **{{XMLExchange}}** tag. In both cases, the service program is then "disconnected" from the session and allowed to run to completion. If the service program subsequently calls **B$ReadRequest** or **B$Exchange**, 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 will run under the current session. This is logically the same as processing a **{{StopService}}** tag in the .SRF file.
BIS-Response-SessionComplete	BIS assumes that the both the service and the session are complete. If the **{{XMLExchange}}** contains an **OnExit**, the **OnExit** is taken immediately and any XML output is ignored. If there is no **OnExit**, the XML output is written to the output stream, completely replacing the **{{XMLExchange}}** tag. In both cases, the service program is "disconnected" from the session and allowed to run to completion. If the service program subsequently calls **B$ReadRequest** or **B$Exchange**, 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 **{{EndSession}}** tag in the .SRF file.
BIS-Response-RecycleService	Reserved for future use.

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	BIS 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.

7.7.1 Notes

- Unlike **B$ReadRequest**, this call returns as soon BIS accepts the output file. This function call does not block waiting for a response from BIS.
- After writing a response, the service program will normally either call **B$ReadRequest** or terminate.
- BIS starts the service timer when this function returns. The program has *ServiceTimeout* seconds to perform one of these actions:
  - Call **B$ReadRequest**
  - Call **B$Exchange**
  - Call **B$SetServiceTimeout**. A parameter of **0** will restart the service timer.
  - Terminate.

  If the service program processes for more than the *ServiceTimeout* interval without performing one of the above functions, BIS 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.

7.8 B$Exchange

This function call is equivalent to calling **B$WriteResponse** immediately followed by **B$ReadRequest**. This function should be used only for the simplest programs because it is not possible to specify the program disposition parameter.

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** will be performed in the absence of a request and 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” on page 50 for a table of result codes and their interpretation.

7.9 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 will wait for a new request. 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 inactivity timer without changing the number of seconds allowed between requests.
- **-1** to reset the timeout value to the default value of **600** seconds (10 minutes).

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 timeout period has elapsed without a request. This function call had no effect.
BIS-Fail-SessionComplete	Service termination is already being requested because a **{{StopService}}** tag was rendered. This function call had no effect.
BIS-Fail-ServiceComplete	Service termination is already being requested because a **{{SessionComplete}}** tag was rendered. This function call had no effect.

7.9.1 Notes

- The default inactivity timeout period is **600** seconds (10 minutes). The section entitled "Session Inactivity Timeout on page 19 describes how the default may be changed for all BIS sessions on this server.

- The inactivity 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 for Windows 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$SetInactivityTimeout()** followed by **B$ReadRequest()**, the updated inactivity timeout interval will 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}}** is not imminent.

7.10 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 to the default value of **30** seconds.

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 **{{StopService}}** tag was rendered. This function call had no effect.
BIS-Fail-ServiceComplete	Service termination is already being requested because a **{{SessionComplete}}** tag was rendered. This function call had no effect.

7.10.1 Notes

- The default service timeout period is **30** seconds. The section entitled "Service Timeouts" on page 19 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 for Windows 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$SetServiceTimeout()** followed by **B$ReadRequest()**, the updated service timeout interval will 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}}** is not imminent.

Appendix A. 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.
APPL_MD_PATH	IIS	Retrieves the metabase path for the **Application** for the ISAPI DLL.
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. 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. 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.

Variable	Platform	Description
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	IIS	The length of the content as given by the client.
CONTENT_TYPE	IIS	The data type of the content. Used with queries that have attached information, such as the HTTP queries **GET**, **POST**, and **PUT**.
GATEWAY_INTERFACE	IIS	The revision of the CGI specification used by the server. The format is **CGI/*revision***. Example: **CGI/1.1**.
HEADER_*HeaderName*	IIS	The value stored in the header ***HeaderName***. Any header other than those listed in this table must be preceded by **"HEADER_**" in order for the **{{Value(*variable*, Server)}}** tag to retrieve its value. This is useful for retrieving custom headers. Unlike **HTTP_*HeaderName***, all characters in **HEADER_*HeaderName*** are interpreted as-is. For example, if you specify **HTTP_MY_HEADER**, the server searches for a request header named **MY_HEADER**.
HTTP_*HeaderName*	IIS	The value stored in the header ***HeaderName***. Any header other than those listed in this table 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	IIS	Returns the value of the **Accept** header.
HTTP_ACCEPT_LANGUAGE	IIS	Returns a string describing the language to use for displaying content.
HTTP_COOKIE	IIS	Returns the cookie string that was included with the request.

Variable	Platform	Description
HTTP_HOST	IIS	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_REFERER	IIS	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.
HTTP_USER_AGENT	IIS	Returns a string describing the browser that sent the request.
HTTPS	IIS	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.
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.

Variable	Platform	Description
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	IIS	Query information stored in the string following the question mark (?) in the HTTP request.
REMOTE_ADDR	IIS	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 have this information, it will set **REMOTE_ADDR** and leave this empty.
REMOTE_IDENT	IIS	If the HTTP server supports RFC 931 identification, then this variable will be set to the remote user name retrieved from the server. Usage of this variable should be limited to logging only.
REMOTE_PORT	IIS	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	IIS	The method used to make the request. For HTTP, this can be **GET**, **HEAD**, **POST**, and so on.
SCRIPT_NAME	IIS	A virtual path to the script being executed. This is used for self-referencing URLs.
SERVER_NAME	IIS	The server's host name, DNS alias, or IP address as it would appear in self-referencing URLs.
SERVER_PORT	IIS	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	IIS	The name and revision of the request information protocol. The format is ***protocol/revision***. Example: **HTTP/1.1**.
SERVER_SOFTWARE	IIS	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**

Variable	Platform	Description
TEMP	IIS	The location of the Windows temporary directory.
TMP	IIS	The location of the Windows temporary directory.
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.

Appendix B. XMLExchange 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 *payload* 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/liantbis/samples/sample1 application with the following data entered into the form fields:

Containing Element	Element	Attribute	Value
bis:content	firstname.Q12		George
	lastname.Q20		Bush
	email.Q33		George.Bush@WhiteHouse.gov
	gender.Q52		Male
bis:cookies	cookie	Name="BISKIT"	iJz3tShlWSNGK2220EUbZWnwR1Xh
bis:query-params	query-param	Name="__xmlexch"	sv3k

The form fields are stored directly into elements contained in the **<bis:content>** element, while the cookies and query parameters are stored as attributed elements into the **<bis:cookies>** and **<bis:query-params>** sections, respectively. Finally, all server variables are output into the **<bis:server-variables>** section (not depicted above). Using the RM/COBOL 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.

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <bis:request xmlns:bis="http://www.xcentrisity.com/2003/bis/request">
- <bis:content>
  <s..LINKARGUMENT />
- <gender.Q52>
- <![CDATA[
Male
  ]]>
  </gender.Q52>
  <s..LINKTARGET />
  <s..idStamp />
- <lastname.Q20>
- <![CDATA[
Bush
  ]]>
  </lastname.Q20>
- <email.Q33__1>
- <![CDATA[
George.Bush@WhiteHouse.gov
  ]]>
  </email.Q33__1>
- <firstname.Q12>
- <![CDATA[
George
  ]]>
  </firstname.Q12>
  </bis:content>
- <bis:cookies>
- <bis:cookie name="BISKIT">
- <![CDATA[
```

```
iJz3tShlWSNGK2220EUbZWnwR1Xh
  ]]>
  </bis:cookie>
  </bis:cookies>
- <bis:query-params>
- <bis:query-param name="__xmlexch">
- <![CDATA[
sv3k
  ]]>
  </bis:query-param>
  </bis:query-params>
- <bis:server-variables>
- <bis:server-variable name="ALL_HTTP">
- <![CDATA[
HTTP_ACCEPT:image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/msword, application/x-shockwave-flash, application/vnd.ms-excel,
application/vnd.ms-powerpoint, */*
HTTP_ACCEPT_LANGUAGE:en-us
HTTP_CONNECTION:Keep-Alive
HTTP_HOST:localhost
HTTP_REFERER:http://localhost/liantbis/samples/sample1/
HTTP_USER_AGENT:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR
1.0.3705; .NET CLR 1.1.4322)
HTTP_COOKIE:BISKIT=iJz3tShlWSNGK2220EUbZWnwR1Xh
HTTP_CONTENT_LENGTH:138
HTTP_CONTENT_TYPE:application/x-www-form-urlencoded
HTTP_ACCEPT_ENCODING:gzip, deflate
HTTP_CACHE_CONTROL:no-cache
  ]]>
  </bis:server-variable>
- <bis:server-variable name="ALL_RAW">
- <![CDATA[
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/msword, application/x-shockwave-flash, application/vnd.ms-excel,
application/vnd.ms-powerpoint, */*
Accept-Language: en-us
Connection: Keep-Alive
Host: localhost
Referer: http://localhost/liantbis/samples/sample1/
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR
1.0.3705; .NET CLR 1.1.4322)
Cookie: BISKIT=iJz3tShlWSNGK2220EUbZWnwR1Xh
Content-Length: 138
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
Cache-Control: no-cache
  ]]>
  </bis:server-variable>
- <bis:server-variable name="APPL_MD_PATH">
- <![CDATA[
/LM/W3SVC/1/root/LiantBIS
  ]]>
  </bis:server-variable>
- <bis:server-variable name="APPL_PHYSICAL_PATH">
- <![CDATA[
D:\Inetpub\wwwroot\LiantBIS\
  ]]>
  </bis:server-variable>
```

```
  <bis:server-variable name="AUTH_PASSWORD" />
  <bis:server-variable name="AUTH_TYPE" />
  <bis:server-variable name="AUTH_USER" />
  <bis:server-variable name="CERT_COOKIE" />
  <bis:server-variable name="CERT_FLAGS" />
  <bis:server-variable name="CERT_ISSUER" />
  <bis:server-variable name="CERT_KEYSIZE" />
  <bis:server-variable name="CERT_SECRETKEYSIZE" />
  <bis:server-variable name="CERT_SERIALNUMBER" />
  <bis:server-variable name="CERT_SERVER_ISSUER" />
  <bis:server-variable name="CERT_SERVER_SUBJECT" />
  <bis:server-variable name="CERT_SUBJECT" />
- <bis:server-variable name="CONTENT_LENGTH">
- <![CDATA[
138
  ]]>
  </bis:server-variable>
- <bis:server-variable name="CONTENT_TYPE">
- <![CDATA[
application/x-www-form-urlencoded
  ]]>
  </bis:server-variable>
- <bis:server-variable name="GATEWAY_INTERFACE">
- <![CDATA[
CGI/1.1
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTP_ACCEPT">
- <![CDATA[
image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/msword,
application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-
powerpoint, */*
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTP_ACCEPT_ENCODING">
- <![CDATA[
gzip, deflate
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTP_ACCEPT_LANGUAGE">
- <![CDATA[
en-us
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTP_CACHE_CONTROL">
- <![CDATA[
no-cache
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTP_CONNECTION">
- <![CDATA[
Keep-Alive
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTP_CONTENT_LENGTH">
- <![CDATA[
```

```
138
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTP_CONTENT_TYPE">
- <![CDATA[
application/x-www-form-urlencoded
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTP_COOKIE">
- <![CDATA[
BISKIT=iJz3tShlWSNGK2220EUbZWnwR1Xh
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTP_HOST">
- <![CDATA[
localhost
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTP_REFERER">
- <![CDATA[
http://localhost/liantbis/samples/sample1/
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTP_URL">
- <![CDATA[
/liantbis/samples/sample1/Default.srf?__xmlexch=sv3k
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTP_USER_AGENT">
- <![CDATA[
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705; .NET CLR
1.1.4322)
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTP_VERSION">
- <![CDATA[
HTTP/1.1
  ]]>
  </bis:server-variable>
- <bis:server-variable name="HTTPS">
- <![CDATA[
off
  ]]>
  </bis:server-variable>
  <bis:server-variable name="HTTPS_KEYSIZE" />
  <bis:server-variable name="HTTPS_SECRETKEYSIZE" />
  <bis:server-variable name="HTTPS_SERVER_ISSUER" />
  <bis:server-variable name="HTTPS_SERVER_SUBJECT" />
- <bis:server-variable name="INSTANCE_ID">
- <![CDATA[
1
  ]]>
  </bis:server-variable>
- <bis:server-variable name="INSTANCE_META_PATH">
- <![CDATA[
/LM/W3SVC/1
```

```
  ]]>
  </bis:server-variable>
- <bis:server-variable name="LOCAL_ADDR">
- <![CDATA[
127.0.0.1
  ]]>
  </bis:server-variable>
  <bis:server-variable name="LOGON_USER" />
- <bis:server-variable name="NUMBER_OF_PROCESSORS">
- <![CDATA[
1
  ]]>
  </bis:server-variable>
- <bis:server-variable name="OS">
- <![CDATA[
Windows_NT
  ]]>
  </bis:server-variable>
- <bis:server-variable name="PATH_INFO">
- <![CDATA[
/liantbis/samples/sample1/Default.srf
  ]]>
  </bis:server-variable>
- <bis:server-variable name="PATH_TRANSLATED">
- <![CDATA[
D:\Inetpub\wwwroot\LiantBIS\samples\sample1\Default.srf
  ]]>
  </bis:server-variable>
- <bis:server-variable name="QUERY_STRING">
- <![CDATA[
__xmlexch=sv3k
  ]]>
  </bis:server-variable>
- <bis:server-variable name="REMOTE_ADDR">
- <![CDATA[
127.0.0.1
  ]]>
  </bis:server-variable>
- <bis:server-variable name="REMOTE_HOST">
- <![CDATA[
127.0.0.1
  ]]>
  </bis:server-variable>
  <bis:server-variable name="REMOTE_USER" />
- <bis:server-variable name="REQUEST_METHOD">
- <![CDATA[
POST
  ]]>
  </bis:server-variable>
- <bis:server-variable name="SCRIPT_NAME">
- <![CDATA[
/liantbis/samples/sample1/Default.srf
  ]]>
  </bis:server-variable>
- <bis:server-variable name="SERVER_NAME">
- <![CDATA[
```

```
localhost
  ]]>
  </bis:server-variable>
- <bis:server-variable name="SERVER_PORT">
- <![CDATA[
80
  ]]>
  </bis:server-variable>
- <bis:server-variable name="SERVER_PORT_SECURE">
- <![CDATA[
0
  ]]>
  </bis:server-variable>
- <bis:server-variable name="SERVER_PROTOCOL">
- <![CDATA[
HTTP/1.1
  ]]>
  </bis:server-variable>
- <bis:server-variable name="SERVER_SOFTWARE">
- <![CDATA[
Microsoft-IIS/5.1
  ]]>
  </bis:server-variable>
- <bis:server-variable name="TEMP">
- <![CDATA[
D:\DOCUME~1\Uwe\LOCALS~1\Temp
  ]]>
  </bis:server-variable>
- <bis:server-variable name="TMP">
- <![CDATA[
D:\DOCUME~1\Uwe\LOCALS~1\Temp
  ]]>
  </bis:server-variable>
- <bis:server-variable name="URL">
- <![CDATA[
/liantbis/samples/sample1/Default.srf
  ]]>
  </bis:server-variable>
  </bis:server-variables>
  </bis:request>
```

Appendix C. 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 "\" should not be used as a pathname separator.

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

Appendix D. Regular Expression Syntax

Regular expressions may be used in the **FIND** and **CHANGE** parameters of the **{{Value}}** tag.

D.1 Metacharacters

This table lists the metacharacters that may be used in **{{Value(...FIND=** ***regexp*****)}}** and **{{Value(...CHANGE=*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**").
^	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 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 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.

Metacharacter	Meaning
`|`	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`".

D.2 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

D.3 Comparison to RM/COBOL LIKE condition regular expressions

- Forms that are common to both Windows expressions and RM/COBOL LIKE expressions are as follows:
  a. Use of “.” for matching any character. There may or may not be a difference here. In Windows expressions, it simply says “any character”, but they probably intended to exclude newline and possibly return. In RM/COBOL LIKE expressions, “.” is actually an abbreviation for the class “[^\r\n]”, that is, any character except newline or return.
  b. Simple forms of class expressions using brackets, with or without negation using the “^” and with sequences specified with a joining “-“.
  c. Repetition operators “?”, “+”, and “*” are the same and have the same effect in both kinds of expressions.
  d. Use of the “\” as an escape for characters that would otherwise be operators and to introduce class abbreviations.
  e. Grouping using parentheses.
  f. Alternatives using “|”.

g. The class abbreviation “\d” for decimal digits is common.

h. The class abbreviation “\n” (newline), but the definitions differ. In Windows expressions, it means “(\r|(\r?\n))”, where “\r” is undefined but is probably 0x0d (return) and “\n” is recursive but in this context is probably simply 0x0a (newline). In RM/COBOL LIKE expressions, “\n” is simply 0x0a (newline).

- Forms in Windows expressions that are not in RM/COBOL LIKE expressions are as follows:

  a. Use of “^” or “$” to match the beginning or end of an expression. RM/COBOL LIKE expressions (from XML Schema) must match the entire string, so these are neither needed nor supported.

  b. The non-greedy operators using “??”, “+?”, and “*?”.

  c. Match groups specified in braces. This form conflicts with the repetition operator in braces in RM/COBOL LIKE expressions.

  d. Use of “\” followed by one or more digits for referencing a previously specified match group value, that is, the value captured by the specified match group.

  e. The negation operator “!”.

  f. The abbreviations “\a” (alphanumeric), “\b” (white space), “\c” (alphabetic), “\h” (hexadecimal digit), “\q” (quoted string), “\w” (simple word), “\z” (integer). Note that “\c” and “\w” are in RM/COBOL LIKE expressions, but have conflicting meanings.

  g. The order of precedence of operators is not described and thus may differ from RM/COBOL LIKE expressions.

- Forms in RM/COBOL LIKE expressions that are **not** in Windows expressions are as follows:

  a. Class expressions, that is, the ability in a class to subtract another class to form a result class that is the difference of two classes.

  b. The repetition operator using braces and counts. This conflicts with match groups in Windows expressions.

  c. Recognition of XML entities such as “&” and character references such as “&#x*h*”, where *h* represents one or more hexadecimal digits, although these may have already been resolved by their appearance in XML pages for BIS purposes.

  d. Regular expression single-character escape sequences (called abbreviations in Windows expressions) “\r” (return) and “\t” (horizontal tab).

  e. Multi-character escapes (called abbreviations in Windows expressions) “\s” (white space), “\S” (not white space), “\i” (initial name characters of XML), “\I” (not initial name characters of XML), “\c” (name characters of XML), “\C” (not name characters of XML), “\w” (all characters except punctuation, separator, symbol and other characters), “\W” (punctuation, separator, symbol and other characters). Note that “\c” and “\w” are in Windows expressions but have conflicting meanings.

  f. Category escapes that match sets of characters based on their Unicode category (“\p{*X*}” and “\P{*X*}”, where *X* represents a Unicode character property designator, for example, L for letters, Lu for uppercase letters, etc., or a Unicode character block, for example, IsBasicLatin.

Appendix E. Log Files

In order to provide usage information over a long period of time, BIS keeps a set of log files in a specific directory. The log files can also be used by the web server administrator or BIS application developer to determine usage patterns of web applications on a BIS server system. The BIS log files consist of variable-length records comprised of space-separated fields. If a field contains spaces or special characters, the field is quoted.

E.1 Log File Location

Under Windows IIS, the log files is written to the BIS application data directory, normally

C:\Documents and Settings\All Users\Application Data\Liant\BIS\LogFiles

Each log file has the following name:

yyyymmdd.log

E.2 Log File Format

Each record begins with a timestamp followed by a record type. The content of the record varies and is dependent on the record type. The general format of each log record is:

```
timeStamp recordType field1 field2 field3…
```

where:

timeStamp	The UTC time when this record was created. The format is **yyyymmddhhmmss** This is a sortable format. Note that the **loggerBegin** record contains a **timeLocal** local time field, and this may be used to determine the UTC offset.
recordType	A single character that encodes the record type. See the table below.
fields	One or more space-separated, variable length fields. • The type and order of the fields varies with **recordType**. • If a field contains spaces, the field will be surrounded by double quotes. • An embedded double quote character is coded with two consecutive double quotes. • A single dash replaces any field with an undefined or unknown value. • Numeric values have leading and trailing insignificant zeroes suppressed.

The following table lists the record types, the log level of that type, and the values that each record of the specified type contains. The value codes are defined below.

Code	Level	Record Type	Fields
L	1	loggerBegin	versionBIS, versionLog, timeLocal
S	2	sessionBegin	idSession, countUses, ipUA, idUA, typeIdUA
V	3	serviceBegin	idSession, countUses, ipUA, idUA, typeIdUA, idService, nameService
R	4	serviceRequest	idSession, countUses, ipUA, idUA, typeIdUA, idService, lengthRequest
r	4	serviceResponse	idSession, countUses, ipUA, idUA, typeIdUA, idService, lengthResponse
v	3	serviceEnd	idSession, countUses, ipUA, idUA, typeIdUA, idService, tallyRequests, tallyLengthReq, tallyLengthResp, timeCPU, histIO
s	2	sessionEnd	idSession, countUses, ipUA, idUA, typeIdUA, tallyRequests, tallyLengthReq, tallyLengthResp, timeCPU, histIO
l	1	loggerEnd	tallyRequests, tallyLengthReq, tallyLengthResp, timeCPU, histIO

where:

Code	A character that indicates the record type.
Record Type	The type of record. Each type is discussed in the table below.
Level	The level of this record type. Future versions of BIS will allow logging to be restricted by level.
Fields	The fields that appear in this type of record. The fields are described in the table below.

E.3 Log Record Types

Here are complete definitions of the various record types

Record Type	Field	Description
loggerBegin	versionBIS	The BIS version number, set to **08.02.00** for the first release
	versionLog	The version number of the log. Set to **1** for the first release
	timeLocal	The local time, in **yyyymmddhhmmss** format

Record Type	Field	Description
sessionBegin	idSession	The cookie value of the session
	countUses	The current use count
	ipUA	The IP address of the user agent in dotted quad notation (normally the same as the HTTP_REMOTE_ADDR server variable)
	idUA	An MD5 digest of user entity identification information (CERT_SUBJECT, AUTH_USER, or - in order of preference, see typeIdUA)
	typeIdUA	A two digit code that indicates the degree of authentication/specificity of the user ID, as follows: 00 No AUTH_USER and no CERT_SUBJECT 10 AUTH_USER and (HTTPS = = off) 20 AUTH_USER and (HTTPS = = on) and no CERT_SUBJECT 30 CERT_SUBJECT and no AUTH_USER 40 CERT_SUBJECT and AUTH_USER
serviceBegin	idSession	The cookie value of the session
	countUses	The current use count
	ipUA	The IP address of the user agent in dotted quad notation (normally the same as the HTTP_REMOTE_ADDR server variable)
	idUA	An MD5 digest of user entity identification information (CERT_SUBJECT, AUTH_USER, or - in order of preference, see typeIdUA)
	typeIdUA	A two digit code that indicates the degree of authentication/specificity of the user ID, as follows: 00 No AUTH_USER and no CERT_SUBJECT 10 AUTH_USER and (HTTPS = = off) 20 AUTH_USER and (HTTPS = = on) and no CERT_SUBJECT 30 CERT_SUBJECT and no AUTH_USER 40 CERT_SUBJECT and AUTH_USER
	idService	ID distinguishing different service instances within the current session
	nameService	Service program name

Record Type	Field	Description
serviceRequest	idSession	The cookie value of the session
	countUses	The current use count
	ipUA	The IP address of the user agent in dotted quad notation (normally the same as the HTTP_REMOTE_ADDR server variable)
	idUA	An MD5 digest of user entity identification information (CERT_SUBJECT, AUTH_USER, or - in order of preference, see typeIdUA)
	typeIdUA	A two digit code that indicates the degree of authentication/specificity of the user ID, as follows: 00 No AUTH_USER and no CERT_SUBJECT 10 AUTH_USER and (HTTPS = = off) 20 AUTH_USER and (HTTPS = = on) and no CERT_SUBJECT 30 CERT_SUBJECT and no AUTH_USER 40 CERT_SUBJECT and AUTH_USER
	idService	ID distinguishing different service instances within the current session
	lengthRequest	Length (in bytes) of the BIS exchange file passed by B$ReadRequest or B$Exchange
serviceResponse	idSession	The cookie value of the session
	countUses	The current use count
	ipUA	The IP address of the user agent in dotted quad notation (normally the same as the HTTP_REMOTE_ADDR server variable)
	idUA	An MD5 digest of user entity identification information (CERT_SUBJECT, AUTH_USER, or - in order of preference, see typeIdUA)
	typeIdUA	A two digit code that indicates the degree of authentication/specificity of the user ID, as follows: 00 No AUTH_USER and no CERT_SUBJECT 10 AUTH_USER and (HTTPS = = off) 20 AUTH_USER and (HTTPS = = on) and no CERT_SUBJECT 30 CERT_SUBJECT and no AUTH_USER 40 CERT_SUBJECT and AUTH_USER
	idService	ID distinguishing different service instances within the current session
	lengthResponse	Length (in bytes) of the BIS exchange file passed by B$WriteResponse or B$Exchange

Record Type	Field	Description
serviceEnd	idSession	The cookie value of the session
	countUses	The current use count
	ipUA	The IP address of the user agent in dotted quad notation (normally the same as the HTTP_REMOTE_ADDR server variable)
	idUA	An MD5 digest of user entity identification information (CERT_SUBJECT, AUTH_USER, or - in order of preference, see typeIdUA)
	typeIdUA	A two digit code that indicates the degree of authentication/specificity of the user ID, as follows: 00 No AUTH_USER and no CERT_SUBJECT 10 AUTH_USER and (HTTPS = = off) 20 AUTH_USER and (HTTPS = = on) and no CERT_SUBJECT 30 CERT_SUBJECT and no AUTH_USER 40 CERT_SUBJECT and AUTH_USER
	idService	ID distinguishing different service instances within the current session
	tallyRequests	The number of requests processed by the service
	tallyLengthReq	Total length (in bytes) of the BIS exchange files passed by B$ReadRequest or B$Exchange
	tallyLengthResp	Total length (in bytes) of the BIS exchange files passed by B$WriteResponse or B$Exchange
	timeCPU	Total CPU time in milliseconds used by this service or – if not available
	histIO	IO request counts used by this service represented by three integers: # of open operations, # of read operations, # of writes/rewrites/delete operations

Record Type	Field	Description
sessionEnd	idSession	The cookie value of the session
	countUses	The current use count
	ipUA	The IP address of the user agent in dotted quad notation (normally the same as the HTTP_REMOTE_ADDR server variable)
	idUA	An MD5 digest of user entity identification information (CERT_SUBJECT, AUTH_USER, or - in order of preference, see typeIdUA)
	typeIdUA	A two digit code that indicates the degree of authentication/specificity of the user ID, as follows: 00 No AUTH_USER and no CERT_SUBJECT 10 AUTH_USER and (HTTPS = = off) 20 AUTH_USER and (HTTPS = = on) and no CERT_SUBJECT 30 CERT_SUBJECT and no AUTH_USER 40 CERT_SUBJECT and AUTH_USER
	tallyRequests	The number of requests processed by the session
	tallyLengthReq	Total length (in bytes) of the BIS exchange file passed by B$ReadRequest or B$Exchange since the start of this session
	tallyLengthResp	Total length (in bytes) of the BIS exchange file passed by B$WriteResponse or B$Exchange since the start of this session
	timeCPU	Total CPU time in milliseconds used by services since the start of this session or – if not available
	histIO	IO request counts accumulated since the start of this session represented by three integers: # of open operations, # of read operations, # of writes/rewrites/delete operations
loggerEnd	tallyRequests	The number of requests processed since the start of this log
	tallyLengthReq	Total length (in bytes) of the BIS exchange file passed by B$ReadRequest or B$Exchange since the start of this log
	tallyLengthResp	Total length (in bytes) of the BIS exchange file passed by B$WriteResponse or B$Exchange since the start of this log
	timeCPU	Total CPU time in milliseconds used by services since the start of this log or – if not available
	histIO	IO request counts accumulated since the start of this log represented by three integers: # of open operations, # of read operations, # of writes/rewrites/delete operations

Appendix F. 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).

- **Symptom**:

```
Liant 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/liantbis/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.
- **Symptom:** one of the following error messages is reported:

```
Cannot create the trace file for session "nH6shZykCtbZmdDZHo0LhJhiVSq5" (the last
attempted filename is "D:\DOCUME~1\Uwe\LOCALS~1\Temp\LiantBIS-nH6s-trace.txt"). The
last error code was 80070005

Cannot reopen the trace file for session "nH6shZykCtbZmdDZHo0LhJhiVSq5" (the last
attempted filename is "D:\DOCUME~1\Uwe\LOCALS~1\Temp\LiantBIS-nH6s-trace.txt"). The
last error code was 80070005

Could not write the trace file to the directory
"D:\DOCUME~1\Uwe\LOCALS~1\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.

Appendix G. Configuration after Installation (Windows)

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 (choose the "Repair" option)
- from the command line

This Appendix describes how to configure the BIS service engine manually from the command line.

G.1 Command Line Configuration

BIS is self-registering. The server registration syntax is

```
LIANTBIS registration-options
```

The registration options are detailed below

/REGSERVER	Registers the BIS and the service engine located in the same directory.
/UNREGSERVER	Unregisters the BIS and the service engine.
/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:

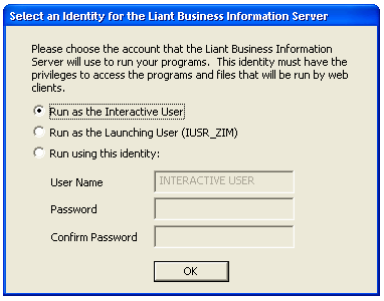
/REGSERVERQ	Quietly registers the BIS and the service engine located in the same directory. No confirmation dialog is displayed.
/REGSERVERO	Only registers the BIS. The service engine's registration remains unchanged. This is useful if you want to locate your service engine system in a directory separate from the BIS.
/REGSERVERQO	Combines the above two options.

The /REGSERVER and /REGSERVERQ options have an additional optional parameter: the pathname of the service engine DLL or the directory containing the server DLL. It is specified like this:

```
/REGSERVER:pathname
/REGSERVERQ:pathname
/REGSERVER: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 service engine in the path.

Note that 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.



The Business Information Server's identity can be configured during installation by specifying options on the command line. If none of

the following options are specified, the server displays the dialog on the right even if **/REGSERVERQ** is specified.

The dialog has three options that determine the context in which BIS will execute. The option can also be selected on the command line, thereby avoiding the dialog.

The options are:

/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 not recommended for deployment.
If the **P** suffix is specified, BIS prompts again 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 again if an error occurs.

/RUNAS:*id*,*pw*
/RUNASP:*id*,*pw*

Runs the server under 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 the file issuing the **/RUNAS** command.
Note: the BIS installation does not presently check that the specified credentials are valid. They are encrypted, stored, and not used until BIS is invoked by the web server.
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 ***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 again if an error occurs.

G.2 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→Adminstrative 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.

3. Right-click Liant Business Information Server 8 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 COBOL 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, enter the name of your anonymous internet account (see below). Click the Add button; make sure Allow is checked next to Launch Permission, then 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.

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 website.

To determine the name of your anonymous internet account:

1. Select Start→Control Panel→Adminstrative 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 RM/COBOL program. This will be RMXML for the sample program. Right-click on 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 website.

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.

G.3 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.

  The changes take effect immediately.

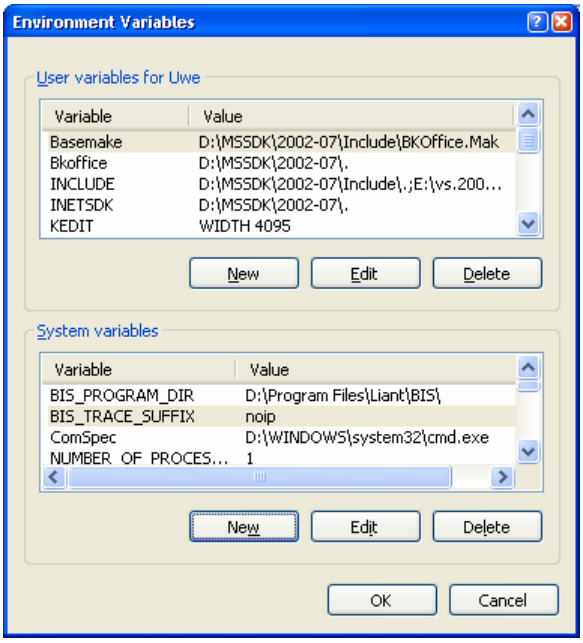


Figure G-1. The Environment Variables Dialog

G.4 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.

G.5 Notes

- Each BIS thread does require system resources, even when idle, so 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 **{{StartService}}**. 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 handler is automatically unloaded after about 20 minutes of inactivity.

- The current setting can be retrieved with **{{Value(MaxThreads, Config)}}**. On UNIX, this always return---s “1”.

Appendix H. Configuration after Installation (UNIX/Apache)

H.1 Configuring Apache

The Apache configuration file for BIS is named **bis.conf** and is included in the Apache server configuration by an **Include** directive placed in the main **httpd.conf** configuration file.

```
<IfModule mod_perl.c>
    Include conf/bis.conf
</IfModule>
```

This isolates all Apache configuration changes for BIS to **bis.conf**, which is described below.

The BIS configuration file contains several sets of Apache configuration directives. The first set of directives creates server variables which are used internally by BIS, and which may also be used in Server Response Files.

```
PerlSetEnv              MASTER_DEBUG            0
PerlSetEnv              STENCIL_DEBUG           0
PerlSetEnv              MESSAGE_DEBUG           0
PerlSetEnv              MAIN_DEBUG              0
PerlSetEnv              LOG_TRACE_FILES         0
PerlSetEnv              KEEP_TRACE_FILES        0
PerlSetEnv              BIS_TRACE_SUFFIX        tag
PerlSetEnv              RUNBIS_TRACE_FILE       /tmp/bistrace
```

These directives affect the amount and location of trace information produced by BIS. The first five directives set a Boolean flag to control trace levels.

MASTER_DEBUG is a master switch that, when set to 0, disables almost all tracing activity. This may be appropriate in a stable production environment. **MASTER_DEBUG** must be set to 1 before most other trace directives will have any effect.

Setting **STENCIL_DEBUG** to 1 causes session level trace messages to appear as BIS processes each of the tags in a Server Response File.

Setting **MESSAGE_DEBUG** to 1 causes session level trace messages to appear as BIS uses IPC (interprocess communications) to coordinate the work of the Apache add-in (which runs as the Apache child user) with the BIS service engine process. Setting **MAIN_DEBUG** to 1 causes internal trace messages to be placed in the file named by the **RUNBIS_TRACE_FILE** directive. These switches can create large amounts of data and should be used only when directed to do so by technical support.

Setting **LOG_TRACE_FILES** to 1 causes session level trace messages to be placed in the Apache error log as the messages are created. This may lead to extremely large error logs and should be used only when directed to do so by technical support.

Setting **KEEP_TRACE_FILES** to 1 causes session level trace message files to be retained after a session is terminated.

BIS_TRACE_SUFFIX is used in conjunction with the **{{TRACE}}** tag described earlier.

```
PerlSetEnv              REFRESH_DIRECTORY       /var/tmp/bishist
```

REFRESH_DIRECTORY names a directory where server responses that may be needed if the client agents (web browsers) request a refresh are stored temporarily. The indicated directory should have permissions which permit create, reading, write and delete access by the Apache child process. If no directory is named, or if this directive is omitted, BIS will not attempt to provide correct responses to refresh requests which will lead to unnecessary session sequence errors.

```
PerlSetEnv              BIS_SESSION_INACTIVITY_TIMEOUT  600
PerlSetEnv              BIS_SERVICE_TIMEOUT     30
```

BIS_SESSION_INACTIVITY_TIMEOUT and **BIS_SERVICE_TIMEOUT** set the default values to be used when establishing a session or when the **DEFAULT** keyword is used in the **{{SessionParms}}** tag. Note, however, that **BIS_SERVICE_TIMEOUT** is usually unnecessary; the **-T** option described below in Service Engine Configuration should be used.

```
#PerlSetVar             BISErrorTextConfig      conf/biserror.conf
#PerlSetVar             BISErrorHTML            conf/BISHTML.conf
```

BISErrorTextConfig and **BISErrorHTML** may be used to configure the error messages displayed in a client browser. Such configuration may be desirable to provide error messages in a national language, or to provide an error display consistent with other parts of a web site. **BISErrorTextConfig** sets the filename of a file which contains lines each of which represent the key and text value of a BIS error message. **BISErrorHTML** sets the filename of an HTML template file which will be used to serve an error page when BIS detects an error that is to be reported to the client browser. Note that, as installed, these configuration directives are comments and must have the '#' character removed from the beginning of each line to become effective. The default behavior is to use the English error message text built in to BIS, and to use the HTML template found at **conf/BISHTML.conf**. Note that the error messages placed in log and trace files on the server may not be configured. Note also that **BISErrorHTML** has no effect on SOAP fault responses.

```
<Files ~ "\.srf$>
        SetHandler      perl-script
        PerlHandler     Apache::Stencil
</Files>
```

This set of Apache configuration directives cause all URIs that request files that end with **.srf** to be processed by the BIS Apache add-in.

```
AddType         text/html .srf
```

This directive causes the default content type of a response for a URI ending with **.srf** to be **text/html**.

```
<Location /BISERROR>
        SetHandler      perl-script
        PerlHandler     Apache::BISError
</Location>
```

This set of Apache configuration directives define the BIS Apache add-in module that will be invoked when BIS determines that an error page, or SOAP error response, must be served.

```
<Location /BISSTATUS>
        SetHandler      perl-script
        PerlHandler     Apache::BISStatus
</Location>
```

This set of Apache configuration directives define an optional BIS Apache add-in module that may be invoked to view the internal status data maintained by the BIS Apache add-in. This data contains the session information and may be useful in diagnosing server problems and evaluating usage. However, since the session data contains information about clients that are using the server, the Apache server administrator should take measures to secure the access to this Location or remove (or 'comment out') these directives in a production server environment.

H.2 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 its command line and its environment. 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/liantbis**. This file is created during the install of BIS.

The command-line options are:

-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.
-A	If specified, the name of the temporary file is passed to the service program in the LINKAGE SECTION, just as if the the A parameter was includedn the **{{StartService}}** tag. This is an obsolete option which will be removed in a future release. The **BIS_FILENAME** environment variable is now used for this purpose.
-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/sysconfig/liantbis**) may be edited. However, the Service Engine does not read this file 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/liantbis**). 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, then starts it again.
condrestart	If the Service Engine is running, stop it, 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.

Appendix I. Windows Security and Authentication

In a Windows Internet Information Server (IIS) environment, the security for your BIS web application and its program (service) and data files is provided by the built-in security mechanisms of IIS. These are based on the Virtual Directory system maintained by IIS and can be manipulated by any user with sufficient Administrator privileges. For this Appendix, Windows Server 2003 is assumed to be the host system, although the procedures for Windows Server 2000 and Windows XP Professional are very similar.

Within the IIS 6.0 Help system, go to Internet Information Services, Server Administration Guide, Security section. There you will find an extensive description of the Windows web security mechanism.

Appendix J. 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 **RM_PROGRAM_DIR** is set to the directory on your machine containing the RM/COBOL development system (with XML Extensions) that you wish to use. This is usually *not* the same directory as the one BIS is installed into. This environment variable may be set by the RM/COBOL installation process, or it might have to be set manually prior to building the sample BIS application.

After verifying and setting **RM_PROGRAM_DIR** if necessary, be sure that a command prompt is present and the current directory is the **src** directory for the sample you are building. At this point the sample may be built by typing (for Windows):

```
build.bat
```

or (for Linux):

```
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.

Appendix K. Glossary

BIS Request Handler – see the BIS Web Server.

BIS Web Server – also referred to as the BIS Request Handler, the BIS components activated when a Stencil (Server Response File) is the target of an HTTP request. The BIS Web Server 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" that 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 RM/COBOL program object file that is invoked by the BIS Web Server and executed by the Service Engine.

Session – a "stateful" sequence of HTTP request/response interactions between a web user agent (e.g., browser) and a BIS Web Server. 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.

Stencil – a file, usually with the extension ".srf" that is used to direct the BIS Web Server in responding to a request. Also referred to as a Server Response File.

URL – a Uniform Resource Locator, the location of a resource on the internet. A URL is a type of URI, and consists of a *scheme* (in this context, HTTP or HTTPS), the name of a *machine*, and a *path* to a resource (e.g., a file). For example, http://liant.com/bis/index.html specifies the file named *index.html* from directory *bis* on server machine *liant.com* using the HTTP scheme. When this is typed into a web browser, the browser issues an HTTP **GET** request on this resource.