Liant Software Corporation

Xcentrisity™
Business Information Server

User's Guide
Version 10

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

Xcentrisity Business Information Server (BIS) User's Guide

Description	Publication Date
Business Information Server (BIS) v8 or later (32-Bit Windows)	February 2004
Business Information Server (BIS) v8 or later (Linux Intel Large File)	May 2004
Business Information Server (BIS) v8 and later (IBM AIX)	August 2005
Business Information Server (BIS) v8.06 and later (SCO Open Server 5.0.7)	October 2006
Business Information Server (BIS) v10 and later (All Systems)	February 2006
Business Information Server (BIS) v10.01 and later (All Systems)	April 2006

Table of Contents

Chapter 1. Introducing the Business Information Server 9

1.1 Overview 9

1.2 Installation on Windows 10
1.2.1 Prerequisites 10
1.2.2 Installation 11
1.2.3 The License Agreement 12
1.2.4 READ ME Information 12
1.2.5 Liant License File 13
1.2.6 User Information 13
1.2.7 Destination Folder 13
1.2.8 Select Features 13
1.2.9 Logon Information 14
1.2.10 Ready to Install 15
1.2.11 Installation Complete 15

1.3 Installation on UNIX 16
1.3.1 Prerequisites 16
1.3.2 Installing Media 16
1.3.3 Installing BIS 17
1.3.4 Configuring Apache 21
1.3.5 Starting Apache and BIS 21

1.4 Testing the Installation 21

1.5 Uninstalling BIS for IIS 22
1.5.1 Removing Only the Web Application Samples 22

1.6 Uninstalling BIS for UNIX 22

Chapter 2. Using BIS 25

2.1 Web Protocols: Requests/Responses 25

2.2 Sessions 26

2.3 Tracking Sessions 27

2.4 Cookies 27

2.5 The Session Root Path and Session Scope 27

2.6 Timeouts 28
2.6.1 Session Inactivity Timeout 28

2.7 Service Timeouts 29

Chapter 3. Server Response Files 31

3.1 Overview 31

3.2 Rendering Tags 31

3.3 The Rendering Process 32
3.3.1 Processing Control Tags 32
3.3.2 Substitution Tags 33

3.4 Tag Options and Parameters 33
3.4.1 Pathnames 33
3.4.2 Referencing Files in System Locations 33
3.4.3 Predefined BIS Environment Variables 34

3.4.4 The RUNPATH Environment Variable 35
3.4.5 Troubleshooting Tags 35

Chapter 4. Tag Reference 37

4.1 The {{Handler}} Tag 37
4.1.1 Notes 37

4.2 The {{ContentType}} Tag 38
4.2.1 Examples 38
4.2.2 Notes 38

4.3 The {{SessionParms}} Tag 38
4.3.1 Notes 41

4.4 The {{StartService}} Tag 41
4.4.1 Accessing the REQUEST from the Service Program 43
4.4.2 Notes 43

4.5 The {{RunPath}} Tag 43
4.5.1 Notes 43

4.6 The {{SetEnv}} Tag 44
4.6.1 Examples 44
4.6.2 Notes 44

4.7 The {{XMLExchange}} Tag 45
4.7.1 Notes 45
4.7.2 Recursive Tag Processing in {{XmlExchange}} 45
4.7.3 The {{FormActionTarget}} Tag in {{XMLExchange}} 45

4.8 The {{StopService}} Tag 46
4.8.1 Notes 46

4.9 The {{SessionComplete}} Tag 46
4.9.1 Notes 46

4.10 The {{Trace}} Tag 47
4.10.1 Notes 48
4.10.2 Examples 49
4.10.3 The {{Trace}} Query Parameter 49
4.10.4 The BIS_TRACE_SUFFIX Environment Variable 49

4.11 The {{TraceDump}} Tag 49
4.11.1 Notes 50

Chapter 5. Conditional Tags and Constructs 51

5.1 The {{If}} / {{Else}} / {{EndIf}} Tags 51
5.1.1 Notes 51

5.2 The {{While}} / {{EndWhile}} Tags 51
5.2.1 Notes 51

Chapter 6. Substitution Tags 53

6.1 The {{Value}} Tag 53
6.1.1 Notes 56
6.1.2 Configuration Variables 56

6.2 The {{ Include }} Tag 57
6.2.1 Notes 57

6.3 Comment Tags 58
6.3.1 Notes 58

Chapter 7. Service Programs 59
7.1 Introduction 59
7.2 Service Program Lifetime 60
7.2.1 ACCEPT and DISPLAY Statements 61
7.2.2 Windows Message Boxes and Dialog Boxes 61
7.3 The XML Exchange File 61
7.3.1 Notes: 62
7.4 BIS Return Codes 62
7.5 Service Program Functions 65
7.5.1 B$ReadRequest 65
7.5.2 B$WriteResponse 67
7.5.3 B$Exchange 69
7.5.4 B$SetInactivityTimeout 70
7.5.5 B$SetServiceTimeout 71
Appendix A. Server Variables Reference 73
Appendix B. XML Exchange Request File Format 83
Appendix C. Windows/UNIX Portability Considerations 91
Appendix D. UNIX BIS 8 Compatibility Issues 93
D.1 Apache Configuration 93
Appendix E. Regular Expression Syntax 95
E.1 Metacharacters 95
E.2 Abbreviations 96
E.3 Comparison to RM/COBOL LIKE Condition Regular Expressions 96
Appendix F. Log Files 99
F.1 Log File Location 99
F.2 Log File Format 99
F.3 Log Record Types 101
Appendix G. BIS Troubleshooting Tips 107
Appendix H. Configurating BIS/IIS after Installation 109
H.1 Command Line Configuration 109
H.1.1 Configuring the Run As Logon ID 110
H.1.2 Retrieving or Changing the Configured Identity 111
H.2 Manual Configuration 113
H.3 Setting Environment Variables 114
H.4 Setting the Maximum Thread Count 115
H.5 Notes 115
Appendix I. Configuration after Installation (UNIX/Apache) 117
I.1 Configuring Apache 117
I.2 Service Engine Configuration 119
I.3 xbisctl Utility 122

I.4 SRC Commands 122
Appendix J. Creating a BIS/IIS Virtual Directory *123*
J.1 Running the BISMkDir Program 123
J.2 Creating the Directory 124
J.3 Testing the New Directory 125
Appendix K. Windows Security and Authentication *127*
Appendix L. Building and Running BIS Samples *129*
Appendix M. Glossary *131*

Chapter 1. Introducing the Business Information Server

1.1 Overview

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

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

Liant BIS has two major components:

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

A **service program** is the COBOL code that is executed by the Service Engine, is application dependent, and not supplied directly by BIS.

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 (usually denoted by a file extension of **.htm** or **.html**), the contents of the file are sent to the browser.

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

Some useful definitions:

User Agent / Client	The program that is used to request information from a server. This program is frequently a web browser, but it could be any program on the user's machine.
HTTP	Hypertext Transport Protocol, a standard encoding scheme used to transmit requests to web servers and receive responses from web servers. HTTPS is a secure version of HTTP.

URL	Uniform Resource Locator, the location of a resource on the internet. A URL consists of a scheme (in this context, HTTP or HTTPS), the name of a machine, and a path to a file. For example, http://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 it 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 (such as BIS) for processing. The two most common web servers are Microsoft's Internet Information Server (IIS), which BIS supports on Windows, and Apache, which BIS supports on UNIX.
Response	A HTTP packet that contains the response to the request. The response may be text, to be displayed in a web browser, or data encapsulated by SOAP for consumption by the requesting program.
Session	Requests are "stateless", that is, the web server processes each request as if it had never received a previous request from the same user agent. A session is a BIS concept that allows sequential requests from the same user agent to be grouped together and preserves state information across requests on the server.

For more definitions, see the Glossary on page 131.

1.2 Installation on Windows

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

1.2.1 Prerequisites

These are the prerequisites for BIS for Microsoft Internet Information Server (IIS) running on Microsoft Windows:

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

- Microsoft Internet Information Server (IIS) must be installed. IIS is the Microsoft web server that listens for HTTP requests on port 80 and HTTPS requests on 443, and routes BIS requests to the BIS Web Server. BIS cannot be installed 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, 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\BIS10\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 two components:

SETUPBIS.EXE	The installation program.
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/IIS 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 shown in Figure 1-1.

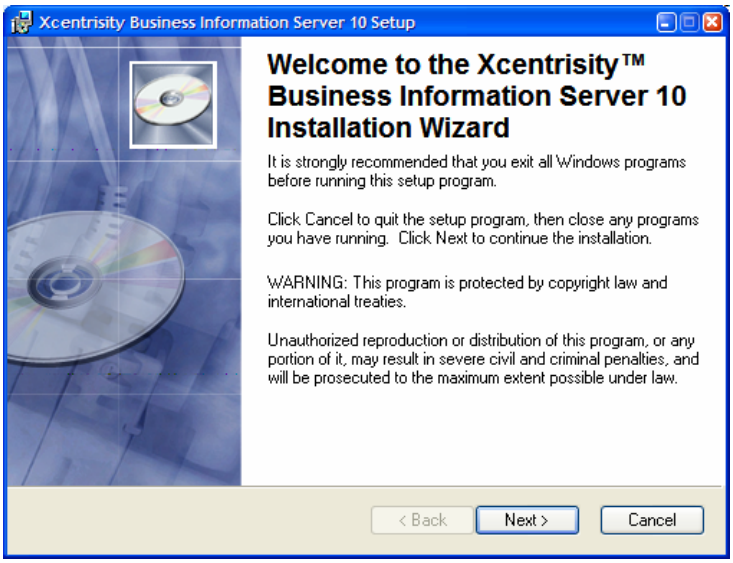


Figure 1-1. Installation Welcome Dialog Box.

- **Note:** In all BIS setup dialog boxes, 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.

- 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 box 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. Press **Ctrl+V** to copy the text from the clipboard into an empty document in *WordPad*.
5. 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 box at the right is not displayed if file **LIANT.LIC** is found in the directory from which the installer was launched.

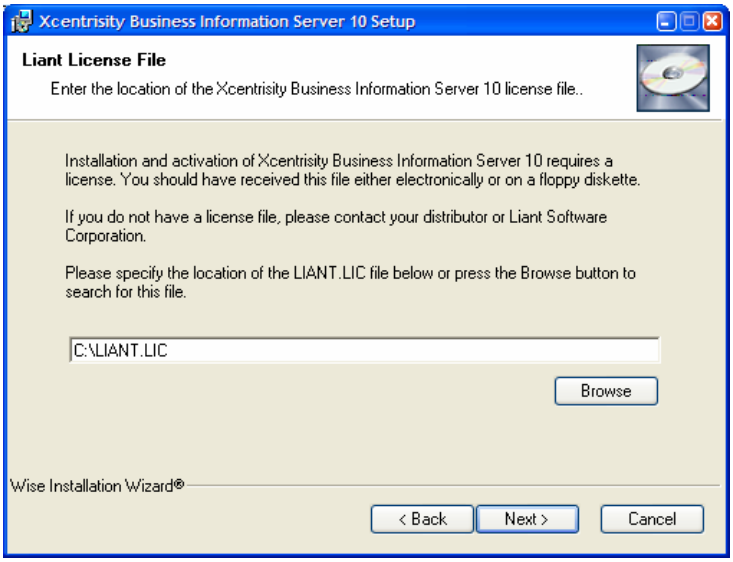


Figure 1-2. Installation License File Dialog Box.

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\BIS10
```

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

1.2.8 Select Features

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

- 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 and includes several sub-features:

        - **Web Applications** are installed in a newly created virtual IIS directory named **XBIS10**. These sample web

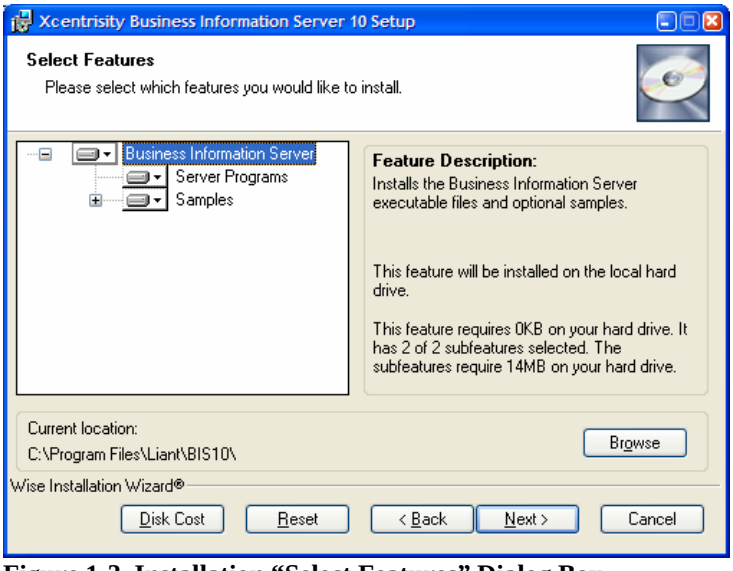


Figure 1-3. Installation "Select Features" Dialog Box.

applications are installed by default because they can be used to quickly verify that the BIS Request Handler is operational. **Note:** Do not change the name of the directory or some of the 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\BIS10\SoapSample3
```

Note: *Microsoft Visual Studio.NET 2003* 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 box 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 box, you must do the following:

- Enter the user name (logon ID) and password that the BIS Service Engine should impersonate when running programs. The installer will validate the user name 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**).

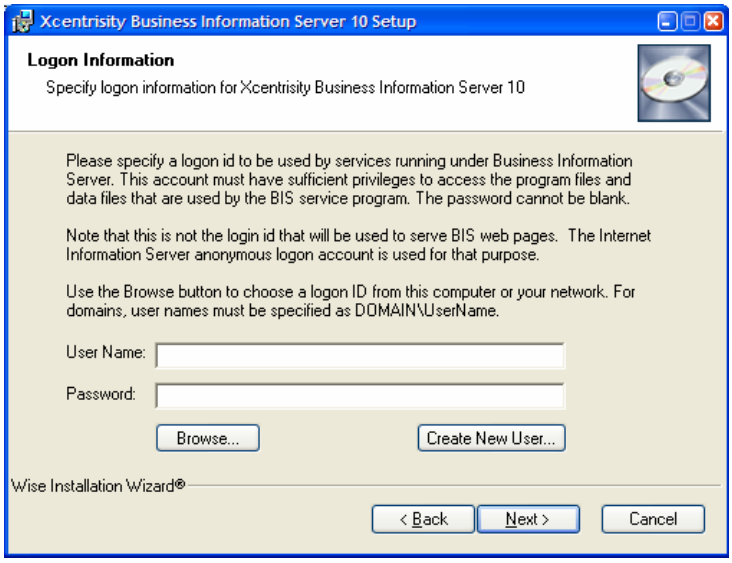
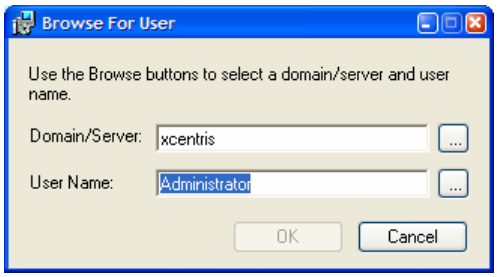
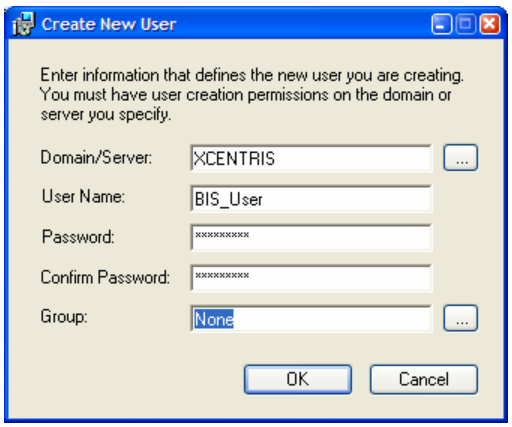


Figure 1-4. The Installation "Logon Information" Dialog Box.



Installation "Browse for User" Dialog Box



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 "Configurating BIS/IIS after Installation" on page 109 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 box 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 107 for assistance.

1.3 Installation on UNIX

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

1.3.1 Prerequisites

BIS on UNIX has the following requirements:

- BIS on UNIX requires a host machine running one of the operating systems below:
  - A host machine running the Linux operating system. BIS has been tested on Red Hat versions 7.3 and 9, Red Hat Enterprise Linux, Fedora Core 4 and 5, SUSE 10.0, and Debian 3.1r1. It should work on any recent Linux release that can support the appropriate version of Apache.
  - A host machine running the AIX operating system version 4.3.3 and above. BIS has been tested on AIX versions 4.3.3 and 5.2 and should work with any newer versions of AIX.
  - A host machine running SCO OpenServer 5 or OpenServer 6 or UnixWare 7.
  - A host machine running Sun Solaris SPARC (2.6, 7, 8 and 9).
- The Apache 2.0 or 2.2 web server must be installed. BIS has been tested on versions 2.0.53, 2.0.55, and 2.2.0. Note that BIS 10 does not support Apache 1.3 and should not be expected to work properly on that platform. Apache normally listens for HTTP requests on port 80 and and for HTTPS requests 443, and when properly configured, routes BIS requests to the BIS Request Handler. BIS will not install unless Apache is already present. On many version of UNIX, Apache is available in a binary format that may be installed from the operating system's installation media or downloaded from the operating system's supplier. In these cases, follow the supplier's instructions for installing Apache. If your system does not have Apache installed, or you wish to download and install the latest version, go to http://httpd.apache.org/docs/2.2/install.html for more information.

1.3.2 Installing Media

BIS for UNIX is available via two sources: ESD (Electronic Software Delivery) and CD-ROM. Both of these sources contain the current instructions for using the media.

- The ESD contains instructions for downloading and unpacking media, and for starting the installation script.
- The CD-ROM contains instructions for mounting the CD-ROM and starting the master CD-ROM installation script.

As such, these instructions will not be repeated here. Depending on your installation source, follow the instructions for installing the media and beginning the installation. At the very least, this will involve logging in as root, changing directory to the directory in which the media was unpacked, or the CD-ROM mount point, and entering the following command:

```
sh ./install
```

1.3.3 Installing BIS

During the installation, the following messages prompts will be presented. For each prompt, the default will be displayed in square brackets and the end of the prompt. Press the <ENTER> key to accept the prompt's default. Otherwise, type the desired value for the prompt and press <ENTER>.

Be sure to select the correct Apache binary such as **/usr/local/apache2** (that is, not one from a source directory).

1.3.3.1 Operating System and User Idenfication

When the UNIX BIS installation script begins, it identifies the operating system and verifies that the current user is root. The following messages are displayed:

```
Operating system detected: Large File Linux (Intel)
Root: Yes [root/0]
```

1.3.3.2 Liant License

The Liant License is displayed using **more** and contains the terms under which you may use this product. Please review these terms carefully and then press Q to terminate the display of the license. The following prompt will then be displayed:

```
Do you accept this license? [y]:
```

Press <ENTER> to signify your acceptance of the terms of the license and the installation will proceed. Enter **N** and press <ENTER> to stop the installation.

1.3.3.3 Locating the Apache Web Server

In order to properly configure the Apache web server and install the BIS Request Handler, it will be necessary to access the web server's installation. Note that this is the directory in which it was installed, not the source directory in which it was built. The installation script will display the following prompt:

```
Use the Apache installed at /usr/local/apache2/bin/httpd? [y]
```

Accept the default if this is the Apache installation in which UNIX BIS is to be installed. Otherwise, enter **N** and the following prompt will be displayed.

```
Specify the directory from which to search for the Apache httpd binary,
or leave blank to search from /.
Apache executable or directory:
```

While it is possible to search the entire disk for the Apache installation, it is faster to just enter the name of the installation directory. For example, entering /usr/local/apache20 results in the following messages:

```
Search [/usr/local/apache20] for httpd binaries.
Use the Apache installed at /usr/local/apache20/bin/httpd? [y]
```

Accepting this installation directory will result in a message being displayed that gives the complete version of the Apache Web Server in which BIS is being installed.

```
Using [Apache/2.0.55]
```

1.3.3.4 Specifying the temporary installation directory

The BIS installation script will need a directory in which it can unpack installation files. This directory may be removed by you following a successful installation. The default for this directory is in the current user's home directory, in a directory named **bis**.

```
Where would you like to copy the installation files?
    [/root/bis]:
```

Specify the desired directory and press <ENTER>.

```
/root/bis does not exist;  create it? [y]:
```

This message is requesting permission to create the directory. Accept the default to give permission to create the directory, or enter **N** to specify a different directory.

If the installation must create the directory, the following message will be displayed:

```
Creating directory /root/bis... Success
```

1.3.3.5 Specifying the installation directory

The BIS installation script will prompt for the directory in which to install the Service Engine and samples.

```
Where would you like to install BIS?
    [/usr/local/liant/bis]:
```

Specify the desired directory and press <ENTER>.

```
/usr/local/liant/bis does not exist;  create it? [y]:
```

This message is requesting permission to create the directory. Accept the default to give permission to create the directory, or enter **N** to specify a different directory.

If the installation must create any the directories, the following message will be displayed:

```
Creating directory /usr/local/liant/bis... Success
Creating directory /usr/local/liant/bis/bin... Success
Creating directory /usr/local/liant/bis/bin/autoload... Success
Creating directory /usr/local/liant/bis/xbis... Success
```

1.3.3.6 Specifying the License Certificate

The BIS installation came with a small file containing the License Certificate, information which signifies that you have permission from Liant to install this product. The installation script will look for the license certificate and silently use it if it can find it. Otherwise, it will prompt for its location:

```
Full name of the license file? [/tmp/liant.lic]
```

Enter the complete pathname of the license certificate in order to proceed with the installation.

1.3.3.7 Configuring the Service Engine options

The installation will display the following prompt to give you the option to modify the default options for the Service Engine's configuration.

```
Do you want to configure BIS Service Engine options? [y]
```

Entering **N** will accept the default options and proceed with the installation. Accepting the default for this prompt will result in the following messages being displayed:

```
Current Service Engine options:
    1 User to run services as?   .   .   .   .    bis
    2 Timezone?  .   .   .   .   .   .   .   .    CST06CDT
    3 Default inactivity timeout, in seconds?     600
    4 Default service timeout, in seconds?   .    30
    5 Maximum number of service processes?   .    100
    6 Maximum number of request handler sessions? 200
    7 Name of temp directory?    .   .   .   .    /var/tmp
    8 Name of log files directory?   .   .   .    /var/tmp/bislogs
    X Done editing the Service Engine options
```

The following prompt will then be displayed:

```
If you would like to change an option, enter its number.  Press Enter to
redisplay the list of options.  Otherwise, enter 'X' to continue [R]:
```

If there is an option that you wish to change, enter its number and press <ENTER>. For example, entering **1** will result in the following prompt:

```
User to run services as? [bis]
```

Enter the new desired value or accept the default. See “Service Engine Configuration” on page 117, for more information about configuring the BIS Service Engine. The prompt requesting the option to change will be displayed again. Enter **R** or just press <ENTER> to review your changes. Enter a number to make more changes. Enter **X** to save your changes and proceed with the installation.

1.3.3.8 Installing BIS Samples

The installation script will display the following prompt:

```
Do you want to install BIS samples? [y]
```

Accept the default to install the BIS samples. Enter **N** to bypass their installation.

1.3.3.9 Specifying the verbosity of the Installation messages

The BIS installation will display the following prompt:

```
Do you want brief install messages? [y]
```

Accept the default to have only summary messages displayed during the installation. Enter **N** to request a more verbose installation.

1.3.3.10 Perform installation

The installation script will now begin the installation. During the install, it may display the following prompt:

```
Warning! /usr/local/apache20/modules/mod_xbis2.so already exists, overwrite
this file? [Y/n/b]
```

Accept the default to overwrite the file. Enter **N** to not overwrite the file and stop the installation. Enter **B** to bypass the installation of this file.

1.3.3.11 Installation complete

At the end of a successful installation, a message similar to the following is displayed:

```
Successfully installed BIS on this system.

Be sure to start the service engine:
```

```
        /etc/init.d/xbisengd start

and to restart Apache:
        /usr/local/apache20/bin/apachectl graceful
```

See the sections below for more instructions on starting the BIS Service Engine and Request Handler.

1.3.4 Configuring Apache

If your version of the Apache installation has a conf.d directory, verify that the mod_xbis.conf configuration file was successfully placed into this directory. If your version of Apache does not use a conf.d directory, verify that the installation edited the main **httpd.conf** configuration file to include the following line:

```
Include conf/mod_xbis.conf
```

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

At this point refer to the appendix chapter “Configuring Apache” on page 117 for more information on configuring the Apache Request Handler.

1.3.5 Starting Apache and BIS

Use the following command to start the BIS service engine on systems other than AIX:

```
/etc/init.d/xbisengd start
```

Use the following command to start the BIS Service Engine on AIX:

```
startsrc -s xbis
```

Use the following command to start or restart the Apache server. Use the actual location of Apache if it is different from this.

```
/usr/local/apache2/bin/apachectl graceful
```

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:

- For BIS installed on a Windows system, click **Start→ All Programs→ Liant→ BIS→ BIS Samples**.

- For BIS installed on either Windows or UNIX, start a web browser and enter the following URL:

```
http://localhost/xbis10/samples/default.srf
```

  Where *localhost* is the name of the Windows or UNIX machine running IIS or Apache. If the web browser is running directly on the same machine as IIS or Apache, then **localhost** may be sufficient.

A web browser should start, and you should see the "Welcome to the BIS Samples" page, as illustrated in Figure 1-5.

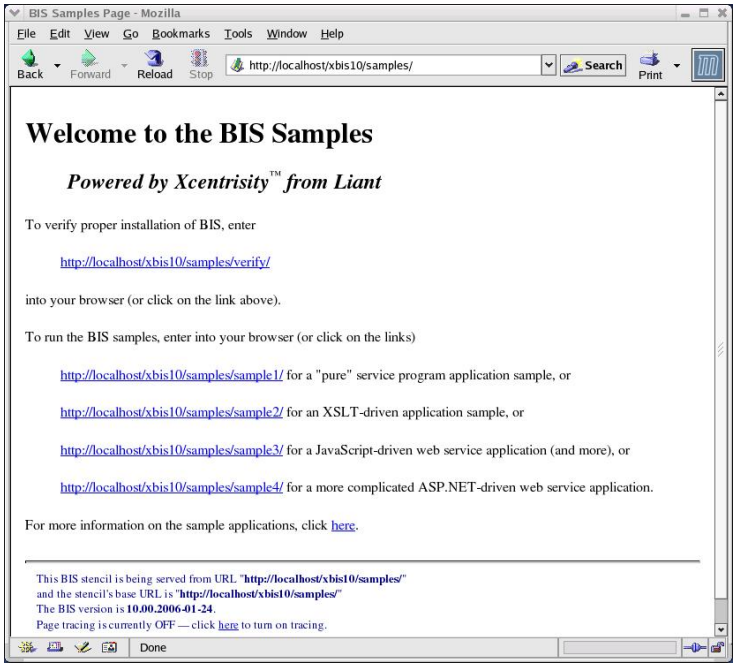


Figure 1-5. The BIS Samples Page.

As an additional test, click on the link to the first sample, **verify**. The BIS Verify sample page will be started. Follow the instructions on this page to complete the verification.

1.5 Uninstalling BIS for IIS

To uninstall BIS/IIS, use the **Add or Remove Programs** control panel applet:

- Click **Start→Control Panel→Add or Remove Programs**.
- Click on **Xcentrisity 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 web site after installation, log onto the server and 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 **XBIS10** and select **Delete** from the popup menu.

1.6 Uninstalling BIS for UNIX

To uninstall BIS for UNIX, log in as root, and perform the following steps:

- Stop the currently running Service Engine by executing one of the two following commands:
  - On AIX: **stopsrc –s xbis**
  - On all other UNIX operating systems: **/etc/init.d/xbisengd stop**
- Change directory to the directory in which BIS was installed: **cd /usr/local/liant/bis**
- Execute the following command: **./uninstall**

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 Web Protocols: Requests/Responses

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

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

There are other methods (HEAD, PUT, 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 method (GET or POST), a URI that specifies the file or resource that is being requested, optional query parameters, and optional form data (if a POST).

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

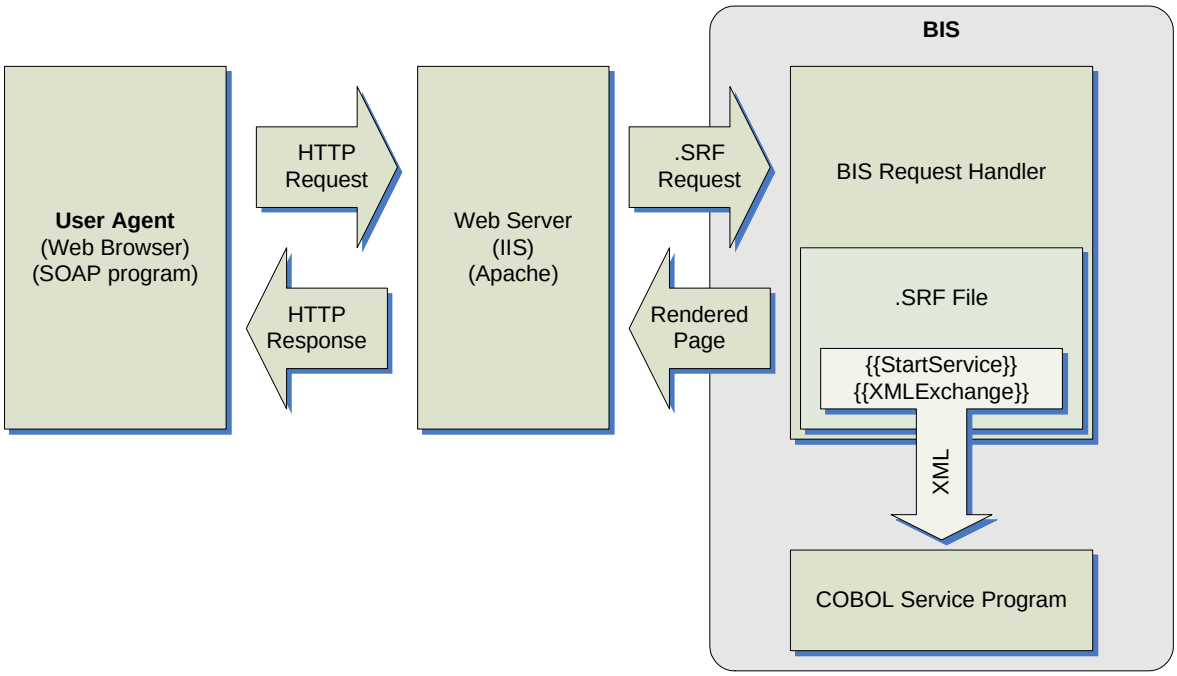


Figure 2-1. BIS Block Diagram.

2.2 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 be routed to the same service program. To make this possible, BIS creates a container on the server called a **Session** when a request first arrives from a particular user agent. The Session contains information that BIS uses to recognize requests as belonging to a sequence, and associates requests with persistent data and services.

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

1. The Session is complete: this can be requested by either the service program or by a special handler tag—the **SessionComplete** tag.

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

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

2.3 Tracking Sessions

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

1. A unique ID may be placed into the URL of subsequent pages.
2. A unique ID may be placed in the query parameter of subsequent pages.
3. The server sends a "cookie" that contains a unique identifier with the response. The user agent saves the cookie, and then includes the cookie with the next request.

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

2.4 Cookies

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

2.5 The Session Root Path and Session Scope

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

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

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

By default, the BIS session cookie uses the path of the URL that caused the session to be created. This can be overridden with the **SessionParms(Path)** tag at the time when the session is created.

Please see "The {{SessionParms}} Tag" on page 38 for more information.

2.6 Timeouts

BIS supports two kinds of timeouts:

- Session Inactivity Timeouts
- Service Timeouts

These timeouts are described in detail in the following sections.

2.6.1 Session Inactivity Timeout

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

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

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

- On the page: embed the **SessionComplete** tag.
- From a service program: call **B$WriteResponse** and specify **BIS-Response-SessionComplete** as the optional parameter.

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

2.6.1.1 Setting the Session Inactivity Time

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

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

  ```
  BIS_SESSION_INACTIVITY_TIMEOUT=600
  ```

  This environment variable sets the timeout to 600 seconds (10 minutes). See "Setting Environment Variables" on page 114 for information about setting and modifying environment variables on

Windows, and “Service Engine Configuration” on page 117 for information on configuring BIS on UNIX.

- The timeout may be set from within a **.srf** file by using the **SessionParms(InactivityTimeout=*seconds*)** tag (see Section 4.3 on page 38). 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 BIS renders an **XMLExchange** tag.

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

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

2.7 Service Timeouts

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

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

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

  ```
  BIS_SERVICE_TIMEOUT=30
  ```

  This environment variable sets the timeout to 30 seconds. See “Setting Environment Variables” on page 114 for information about setting and modifying environment variables on Windows and “Service Engine Configuration” on page 117 for information on configuring BIS on UNIX.

- The timeout may be set from within a **.srf** file by using the **SessionParms(ServiceTimeout=*seconds*)** tag. Note that this parameter is specified in seconds and takes effect as soon as the tag is rendered.

- The service program may set the timeout with the **B$SetServiceTimeout** call. Calling this function with a parameter of **0** restarts the timer without changing the current value. This is useful as a “keep-alive” function when performing lengthy processing.

Of the above, the **BIS_SERVICE_TIMEOUT** variable and **ServiceTimeout** option keyword have the lowest priority and are overridden by either **SessionParms tag** or **the B$SetServiceTimeout** call.

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 from a User Agent.

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 4096 characters.

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

- Turn off line and word wrapping in your HTML editor for **.srf** files. Note that Visual Studio 2003 and 2005 properly handle 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.

3.3 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.

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.

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

3.3.1 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.

3.3.2 Substitution Tags

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

3.4 Tag Options and Parameters

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

3.4.1 Pathnames

There are two kinds of pathnames used within tags:

- A fully qualified pathname begins with a slash. On BIS/IIS, the slash may optionally be preceded by a drive letter specification.

- A relative pathname is any pathname that does not follow the above rules.

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

The current directory for the BIS Service Engine is 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 that **.srf** file.

On BIS/IIS, 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 service programs in conjunction with the **SetEnv** tag.

3.4.2 Referencing Files in System Locations

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

Under BIS/IIS, the following environment variables are useful in pathnames. Note that **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 BIS/IIS, 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" by specifying a path of "@MyPrograms/abc.cob". See "Setting Environment Variables" on page 114 for information about setting and modifying environment variables on Windows.

On UNIX, use the x**bis.conf** configuration file to define BIS environment variables. See "Configuring Apache" on page 117 for details.

3.4.3 Predefined BIS Environment Variables

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

Variable	Description
BIS_PROGRAM_DIR	The directory from which the BIS Service Engine is loaded. On Windows, this will normally be the XBIS.EXE program in "c:\program files\Liant\BIS10".
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 response. • 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 POSTed-back form variables, the request variables, and server variables, all encoded as an XML document. By default, this file is created in the Windows TEMP or the UNIX /tmp directory. Both the BIS Request Handler and the Service Engine must have permission to create, read, and write files in this directory. The BIS installation procedure adds the required permissions to this directory.

3.4.4 The RUNPATH Environment Variable

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 program 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.

3.4.5 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 itself visible in your web browser?

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

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

     The trace files use the following naming convention:

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

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

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

     Note that on UNIX, the **BISMasterTrace** configuration option must be enabled before any tracing can occur. See "Configuring Apache" on page 117, 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 indicates that this particular **.srf** file contains Xcentrisity BIS rendering tags.

```
handler *
```

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

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. (Note that BIS/IIS allows **include** tags to precede the **handler** tag.)

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

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**". On BIS/Apache, if the content type of the request message indicates an XML message, the default content type of the response is "**text/xml; charset=utf-8**".

- If **{{ContentType}}** is specified multiple times on a page, the last instance is used.

4.3 The {{SessionParms}} Tag

This tag allows various session attributes to be set:

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

where:

InactivityTimeout	Determines how long a session survives without user interaction.	
	DEFAULT	Resets the timeout to the default setting: 600 seconds (10 minutes).

	seconds	An integer that specifies the number of seconds before the session terminates. The minimum value is 10 seconds (useful for testing) and the maximum value is 1,000,000 seconds (about 11 days).
ServiceTimeout	Determines the maximum length of time the Service Program may run when a request is received.	
	DEFAULT	Resets the timeout to the default setting: 30 seconds.
	seconds	An integer that specifies the number of seconds before Service Program termination processing begins. The minimum value is 10 seconds (useful for testing) and the maximum value is 3,600 seconds (one hour).
Path	Specifies the root path of the current session. This parameter is ignored unless the page being served is the initial session page.	
	DEFAULT	The session root path is set to the path of the current request. See “The Session Root Path and Session Scope” on page 27 for more details.

	path	Explicitly sets the session root path to ***path***. The ***path*** may be specified as a relative or an absolute URL path and must specify a path segment contained in the URL path of the initial page. In addition, under IIS, the specified ***path*** must contain at least as many path segments as the application root path (the base directory for the BIS application)—that is, the path cannot be closer to the root of the web than the BIS virtual directory. For example, if the requested page is **http://liant.com/xbis/apps/states/texas/default.srf** the default session path will be **/xbis/apps/states/texas** These paths may be specified to set the session root path to **xbis/apps**: **Path="/xbis/apps"** **Path=../../** These paths set the default session path to the directory containing the requested object: **Path=DEFAULT** **Path=.** **Path="/xbis/apps/states/texas"** These paths are invalid and will be reported as being invalid in the trace file: **Path=/xbis/apps/states/california** **Path=../florida** These directories are not contained in the path. In addition, for IIS servers, the path cannot be closer to the root than the application base path (the path that describes the virtual directory that contains the BIS server).
Scope	Determines the scope of the current session. This parameter may be specified at any time and is not inherited by new sessions.	
	ALL	All pages served from the session base directory and subdirectories of the session base directory are served as part of the current session. This option is the initial default for all new sessions and is appropriate for most applications.

	ISOLATE	Only pages served from the session base directory are included in the current session. A new, non-isolated session is started when a page is requested from a subdirectory of the session base directory. The **ISOLATE** option allows a single user agent to use more than one BIS session as long as the sessions are based in separate directories on the server.

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

  http://liant.com/xbis/apps/states/default.srf

  and the application contains a link to this page, located in a more specific directory:

  http://liant.com/xbis/apps/states/texas/default.srf

  If **SessionParms(Scope=All)** is in effect, the subordinate page will be served from the same session as the initial page. However, if **SessionParms(Scope=Isolate)** is in effect, a new session will be created for the subordinate page.
- For a description of the proper usage of the session base and session scope options, see “The Session Root Path and Session Scope” on page 27 for more details.

4.4 The {{StartService}} Tag

This 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 between ***program*** and the first comma or closing parenthesis is interpreted as a Service Engine option and is passed to the Service Engine without further interpretation. Some useful options include:	
	-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**	Specify 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 on UNIX, BIS 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 program to be active in a session. Note the following:

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

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

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

Examples:

```
1. {{ StartService ( myapp ) }}
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 RM/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 RM/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 multiple **StartService** tags. An additional **StartService** tag is ignored if it specifies a service that is already running. If it specifies a difference service, the service started by the previous tag is stopped before the new service is started.
- The **StartService** tag must precede any tags that depend on the service program being active. Such tags currently include **XMLExchange**.

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, which will replace the directory list set by the previous **RunPath** tag. 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 "." (that is, "current directory") to the path.

- See "Setting Environment Variables" on page 114 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 Request Handler's environment. The **Value(variable,ENV)** tag will **not** retrieve variables set by this tag.

- Multiple **SetEnv** tags may be specified in a **.srf** file and are processed in the order in which they occur. Note that these tags must precede the **StartService** tag.

- On BIS/IIS, the scope of a **SetEnv** tag is the current request, not the current session. On BIS/Apache, the scope of the **SetEnv** tag is the current session.

- To unset an environment variable, omit the **"=value**". Note that an unset variable is different from a variable that has a blank (or empty) value.

- All characters to the right of the equal sign up to the first space before the right-most parenthesis are stored as the value. If the value is quoted as in the example above, quotes will also be set in the environment.

4.7 The {{XMLExchange}} Tag

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

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

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

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.

- On BIS/IIS, the **OnExit** in the first **XMLExchange** tag following a **StartService** tag is ignored. This allows any service startup errors to be reported and corrected.

4.7.2 Recursive Tag Processing in {{XmlExchange}}

Beginning in version 10, BIS recursively processes tags in the service program's response output, as if the response output was a stencil. Tag substitution occurs as the service output is written to the response page (replacing the **{{XmlExchange}}** tag), and substitution is performed in the context of the page that contains the **{{XmlExchange}}** tag.

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

The **{{FormActionTarget}}** tag discussed in the next section is a tag that is specifically intended to be included in generated **{{XmlExchange}}** output. Note that BIS version 8 processed the **{{FormActionTarget}}** tag as a special case, but BIS version 10 allows any tag to be embedded in the output—even another **{{XmlExchange}}**.

4.7.3 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** tag immediately after a **StopService** tag is invalid and, if present, the **OnExit** parameter in that tag will be processed.

- The **StopService** tag may be immediately followed by a **StartService** tag. In this case, a new service program is started. Once the **StopService** tag is rendered, the service program is considered terminated even if it needs a few additional seconds to actually stop.

- This tag is ignored if there is no service program attached to this session.

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 )
```

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

START **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**.

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

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

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

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

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

QUERYPARAM=*value* **QP=*value***	**QUERYPARAM** and **QP** are synonymous and select a query parameter that can be used to dynamically specify the options above. 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.
NOQUERYPARAM **NOQP**	Disable the query parameter set by **QUERYPARAM** or **QP**.
IP=xx.xx.xx.xx **[-x.xx.xx.xx]**	**IP** allows trace output to be restricted to requests originating at one or more IP addresses. If an IP restriction is in effect, trace output is restricted exclusively to requests from those particular IP addresses. A comma-separated list of IP addresses or ranges may be specified. The list of IP restrictors is processed from left to right. Note that specifying **127.0.0.1** will allow access from a web browser running on the host's console. In this case, access the pages using **localhost** as the name of the host.
NOIP	Disables restricting IP addresses.

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 (that is, **PAGE**, **FILE**, **TAG, EXCHFILES**) are specified.

- The trace mode is part of the session and is "sticky". This means that the trace setting persists in the session until it is changed by either another **trace** tag or a query parameter (if enabled). So if you have more than one page in your application, the **trace** tag is required only on your initial page.

- Only **.srf** files may be traced. If you follow a link to an **.htm** or **.asp** page, those pages will not be traced. If those pages link back to a **.srf** file in this application's virtual directory, then tracing will once again resume as long as the session is still active.

- Be cautious when enabling tracing in a way that exposes the trace information to site visitors. Trace information will reveal some information about your system that may be useful to

intruders. The **QUERYPARAM** is configurable to help secure your web by allowing tracing to be turned on and off using a keyword that is not easily guessed by intruders.

4.10.2 Examples

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

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

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

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

For example, if your trace statements look like this:

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

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

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

- This will effectively append these parameters to every **Trace** tag executed on the server without requiring the actual **.srf** file to be edited. Note that the **.srf** files must contain a **Trace** tag for this feature to take effect.
- See “Setting Environment Variables” on page 114 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 (that is, removed from the output) if tracing is not being performed.
- Because trace information is accumulated as the page is rendered, it is most useful for the **TraceDump** tag to be specified near the end of the page.
- If this tag is omitted and tracing is enabled, BIS/IIS 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 tag currently supported in the **If** tag, and the **Value** tag must contain a **MATCH** operator.
- 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 tag currently supported in the **While** tag, and the **Value** tag must contain a **MATCH** operator.
- 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 after 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|"variable" [, source] [,operations]…)
```

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

V10 On BIS/IIS, if **variable** is enclosed in quotes, the variable name is treated as a literal string and is not resolved further unless one of the **source** options below is specified.

V10 On BIS/Apache, if **variable** begins with a quote, it is treated as a literal and no **source** option is permitted.

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

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

SERVER	Specifies that **variable** is a server variable. This is the default if none of the other sources below are specified and if the string is not quoted. Under BIS/Apache, **ENV** and **SERVER** are identical.
CONFIG	Specifies that **variable** is a special server configuration value. A list of **CONFIG** variables appears at the end of this section.
COOKIE	Specifies that **variable** is a cookie.
ENV	Specifies that **variable** is an environment variable instead of a server variable. Note that, on BIS/Apache, **ENV** and **SERVER** are identical. See "Setting Environment Variables" on page 114 for information about setting and modifying environment variables on Windows.
FORM	Specifies that **variable** is a <form> variable.
QUERYPARAM **QP**	Specifies that **variable** is a URL query parameter. **QP** is accepted as an alias for **QUERYPARAM**.

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

DEFAULT=*value*	Specifies a default value for **variable** if the **variable** is empty. This option has no effect unless the **variable** is empty at the point the **DEFAULT** operation is performed. • If the **variable** is empty when the end of the operations list is reached, the Value tag is simply removed from the output stream. • If **DEFAULT** is specified, the tag is replaced by ***value***. If there are additional operations to the right of the **DEFAULT** operation (that is, **GETDIR**, **TOUPPER, URLENCODE**), these are performed on the defaulted **value.**
GETDIR	Same as **GETPATH** (see below), except only the directory portion of the pathname is extracted. This is the part of the pathname that follows the scheme and hostname up to the last slash in the pathname. Note that if **variable** is a pathname that contains a drive letter, the drive letter is also returned. The extracted pathname never ends in a slash.
GETQUERYSTRING V10	If **variable** is a URL, returns the query string. If not present or if **variable** is a pathname, an empty string is returned.
GETHOST V10	If **variable** is a URL, extracts the hostname. If **variable** is a pathname, or no host name is present, an empty string is returned.
GETNAME	Same as **GETPATH**, except only the filename portion of the pathname is extracted. This is the part of the pathname that follows the last slash but excludes the **#fragment**, **?querystring**, and **;parameters**.
GETPATH V10	If **variable** is a URL, extracts the path portion of the URL. This is the portion of the URL that excludes the scheme, the hostname, and the query string. If variable is a pathname, it is unchanged.
GETSCHEME V10	If **variable** is a URL, extracts the scheme. This will normally be **http** or **https** without the terminating colon or slashes. If **variable** is a pathname or a URL without a scheme, an empty string is returned.
SUBSTITUTE=*/pattern/replacement/* **SUB=*/pattern/replacement/***	Allows you to substitute all occurrences of *pattern* in the value with a replacement pattern. The operation is performed on the current value after all transforms to the left have been performed. Processing continues with the modified value. **SUB** is accepted in place of **SUBSTITUTE** for brevity. Both *pattern* and *replacement* are regular expressions. (For more information, see “Regular Expression Syntax” on page 95).
TOUPPER	Converts the value to all upper-case characters. Equivalent to **SUBSTITUTE="/.*/\U&/"**.
TOLOWER	Converts the value to all lower-case characters. Equivalent to **SUBSTITUTE="/.*/\L&/"**.
URLDECODE	Decodes a string that has been URL-encoded. This is primarily useful when retrieving a server variable.

URLENCODE	Encodes a string for reliable HTTP transmission from the web server to a client as a URL. For example, **"This is a <Test String>."** will be encoded as: **"This%20is%20a%20%3cTest%20String%3e."**
HTMLDECODE	Decodes a string that has been HTML-encoded. This is primarily useful when retrieving a server variable.
HTMLENCODE	Encodes a string for reliable HTTP transmission from the web server to a client as HTML. For example, **"This is a <Test String>."** will be encoded as: **"This is a &ltTest String&gt."**
MAKEABS V10	Assumes that the string is a relative URL, and makes the URL absolute, using the location of the stencil that was requested by the client as the base URL (see **REQUEST_URL** in "Configuration Variables" for details). If the string is not a URL, it is not altered. If the input string is an absolute URL, it is cleaned up (that is, redundancies such as dir/../ are removed) but is otherwise unchanged. See RFC 3986 for details about how relative URLs are resolved by this operation.

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

MATCH=*regexp*	Applies the regular expression against the current value and returns true if it matches and false if it does not match but does not return any text for rendering. This allows **Value** to be used in **If** tags. See Appendix D, "**Error! Not a valid result for table.**" beginning on page 95.

For example, the tag:

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

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

On Windows XP, the tag:

```
{{ 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 **MATCH** option, the condition is **TRUE** if **Value** evaluates to a non-empty string; otherwise, **FALSE**.

- Regular expressions must be delimited. The first nonblank character after the '=' is the delimiter for the regular expression. The expression begins at the character following the delimiter and extends up to, but not including the next occurrence of that character.

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

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

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

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

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

```
{{ VALUE(userid, QUERYPARAM, MATCH="fred\s", URLENCODE) }}
```

- Commas cannot occur inside delimited or quoted strings because commas always separate parameters. If a comma is required, use "%2c" and URLDECODE the string to convert the "%2c" to a comma.

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. However, this server variable may be undefined under Apache.
MAXTHREADS IIS	Resolves to the number of threads configured in the BIS thread pool. This is the number of threads that are available for requests. Under Apache, this is undefined (use **DEFAULT=1** if portability is desired).
REQUEST_URL V10	Retrieves the completely qualified URL of the stencil (**.srf**) file that was requested by the client. This includes the scheme, hostname, port number (if non-standard), path, and parameters, query string, and fragment (if specified).

REQUEST_BASE_URL V10	Retrieves the completely qualified base URL of the current stencil (**.srf**) file that was requested by the client. This includes the scheme, hostname, port number (if non-standard), and the path (which always ends in a slash). The base URL is defined as the directory that contains the stencil that was requested by the client.
STARTSERVICE	Returns the entire argument list of the currently active **StartService** tag, including commas. If there is no active service program, the value is considered undefined and may be overridden with the **DEFAULT** operation.
SCHEME V10	Returns the scheme that was used to request the current page: currently returns either **http** or **https**. Note that the **://** delimiter that follows the scheme is not included. This is useful for constructing URLs: **<a href="{{ Value(SCHEME, CONFIG) }}:// {{ Value(HTTP_HOST,URLENCODE) }} {{ Value(HTTP_URL,GETDIR,URLDECODE,URLENCODE) }}/default.srf">** (Note that the above must be on a single line, or spaces will be inserted.) Under BIS/IIS, the scheme is derived from the **SERVER_PORT_SECURE** server variable, where a value of 0 indicates http and nonzero indicates https.
SERVICENAME	Retrieves the name of the currently active service program. If there is no active service program, the value is considered undefined and may be overridden with the **DEFAULT** operation.
VERSION	Retrieves BIS version information. The format of the version number is **aa.bb.cc.yyyy/mm/dd**, where **aa.bb.cc** indicates the numeric major/minor/patch level version, and **yyyy/mm/dd** is the build date.

6.2 The {{ Include }} Tag

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

```
include filepath
```

Where ***filepath*** is the path to the file whose contents 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 so tags can 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. Note that any included **.srf** file must also contain a **Handler** tag.

- Relative pathnames in ***filepath*** are interpreted as relative to the location of the **.srf** file that contains the **include** tag.

- If an included **.srf** file contains a **StartService** tag, the service program's working directory is the directory that contains the **.srf** file that rendered the tag.

- The included file does not need to be a **.srf** file. For example, an **.html** file, a **.css** (cascading style sheet) file, or a **.js** (JavaScript) file can also be included, and in this case, the **Include** tag is simply replaced by the contents of the specified file.

- On BIS/IIS, an **include** tag can appear anywhere in a **.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 the following:

```
{{//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** tag is replaced by the XML produced by the COBOL program. However, the new line character that follows each of these tags would remain in the output, resulting in 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 newline characters. The **XMLExchange** tag is not followed by a newline-consuming comment because a newline is desirable before the end of the file and, in this case, the emitted XML does not contain any newline characters.

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.

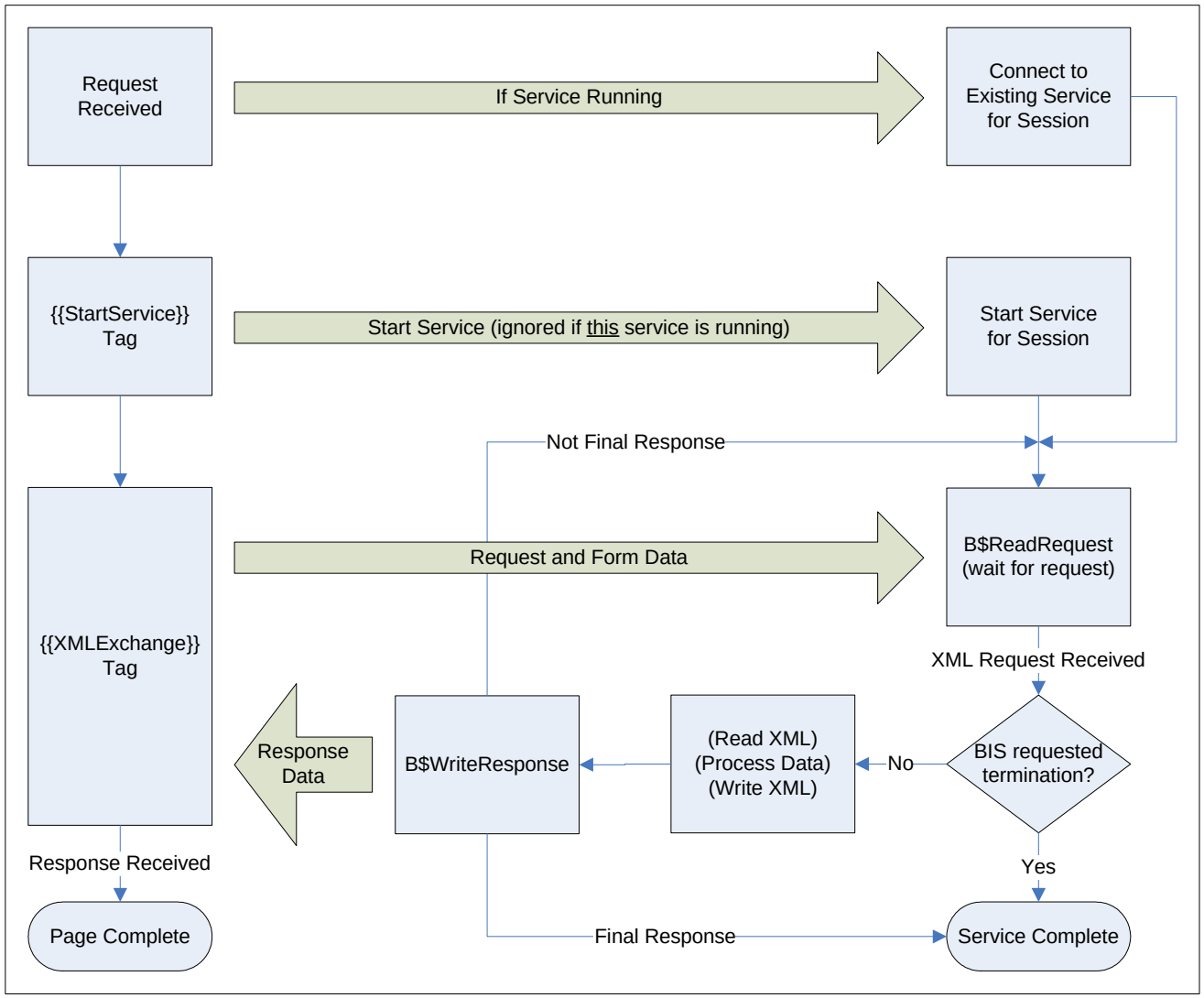


Figure 7-1. BIS Control Flow.

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 the BIS Request Handler is ready to provide a request. To avoid deadlocks, once BIS begins to process the **XMLExchange** tag:

- The service program must call either the **B$ReadRequest** or **B$Exchange** function within *ServiceTimeout* seconds.
- Alternatively, the program may request additional time by calling **B$SetServiceTimeout** using 0 to reset the timer.

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

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

- Wait for the next request for the current session by calling **B$ReadRequest** or **B$Exchange**.
- Terminate (for example, with **STOP RUN**.)

If neither of the above two events occurs, the BIS Service Engine will terminate the service program.

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.

The following general rules apply to service programs:

- A given BIS session may have only one active service program at any time.

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

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

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 clients.

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.

If a program does become unresponsive because it is looping, awaiting a response to an ACCEPT, BIS will terminate the program after the service timeout interval expires as described above unless a call to **B$ReadRequest** or **B$SetServiceTimeout** in included in the loop. Also, to avoid the creation of very large trace files, BIS/IIS will eventually suspend the tracing of **DISPLAY** statements for the remainder of the request.

7.2.2 Windows Message Boxes and Dialog Boxes

Because the service program does not have access to the Windows desktop, it is not appropriate to display a message box or a dialog box. If the service program did attempt to interact with the user in this way, it will suspend waiting for a response that cannot ever come. 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. The filename, therefore, is accessible to the RM/COBOL program via the **C$GetEnv** function:

```
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.
```

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

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

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

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

```
bisiiiiiiiiiiiiiiiiiiii-ssss.xml
```

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

7.3.1 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$Exchange** to both receive requests and write responses.
- While the filename is determined when the service engine is started, the file itself is not created until **B$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. Note that the severity of an error condition increases with the value of the return code.

00-09	Success! For **B$ReadRequest** and **B$Exchange**, the request data is available in the XML exchange file. For **B$WriteResponse**, the response was accepted by the Request Handler.	
	00	**BIS-Success**
		The data transfer succeeded and the XML file contains the result of the operation.
10-19	A non-fatal event occurred, and recovery is possible. While no such conditions currently exist, these codes are reserved for future use.	
20-29	A failure occurred but the program should be able to recover. The XML file was not updated.	
	20	**BIS-Warn-RequestTimeExpired**
		A timeout parameter was specified on the **B$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 attempt 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 or possible planned session/service expiration event occurred and the program cannot continue. The XML file was not updated and the program should terminate as soon as possible or BIS will terminate the program after the service timeout interval expires.	
	50	**BIS-Fail-SessionAbandoned**
		The session timed out.
	51	**BIS-Fail-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—a **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 Request Handler.
	88	**BIS-Fail-ServerInternalError**
		An internal error occurred while the service program was communicating with the BIS Request Handler.
	89	**BIS-Fail-ServerMemoryManagement**
		A memory management failure occurred in the BIS service program.
	90	**BIS-Fail-ServerBadMessage**
		An internal error occurred while the service program was communicating with the BIS Request Handler.

	91	**BIS-Fail-ServerBadLength**
		An internal error occurred while the service program was communicating with the BIS Request Handler.
	92	**BIS-Fail-ServerBadParameter**
		An internal error occurred while the service program was communicating with the BIS Request Handler.
	93	**BIS-Fail-ServerWrongMsg**
		An internal error occurred while the service program was communicating with the BIS Request Handler.
	99	**BIS-Fail-ServerConnectionLost**
		The connection between the BIS service program and the BIS Request Handler failed.

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.

7.5.1 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
The BIS Request Handler renders an **XMLExchange** tag for the current session	This tag causes the current request data to be encoded into XML and placed into the file specified by the **BIS_FILENAME** environment variable.
The BIS Request Handler renders a **StopService** or **SessionComplete** tag for	This indicates that the service is no longer required. The service program should terminate and is granted *ServiceTimeout* seconds to do so.

the current session	
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 are as follows (see "BIS Return Codes" on page 62 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 Appendix B, "XML Exchange Request File Format" beginning on page 83.

7.5.1.1 Notes

- The BIS Service Engine starts the service timer when this function returns. The program is then given *ServiceTimeout* seconds to process the request and perform one of these actions:

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

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

- If 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).

7.5.2 B$WriteResponse

This function call transmits a response to the BIS Request Handler 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 may be used to indicate 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 are as follows (see “BIS Return Codes” on page 62 for a complete table):

BIS-Status Code	Event Description
BIS-Response-Normal	The default is that BIS makes no assumptions about what the service program will do next. However, the service program is granted only 30 seconds to exit or to read the next request.
BIS-Response-ServiceComplete	Indicates that the service has completed processing, cannot accept any additional requests, and is about to terminate. The XML output is written to the output stream, completely replacing the **XMLExchange** tag. The service program is then disconnected from the session and allowed to run to completion in the background. If the service program subsequently calls **B$ReadRequest** or **B$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	Indicates that the both the service and the session are complete and cannot not accept any additional requests. The XML output is written to the output stream, completely replacing the **XMLExchange** tag. The service program is disconnected from the session and allowed to run to completion in the background. If the service program subsequently calls **B$ReadRequest** or **B$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 **StopService** 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	The BIS Request Handler accepted the response. This does not mean that it was delivered to the user agent. However, the service program should presume success and resume processing.
BIS-Warn-ResponseUnexpected	There is no pending request to respond to.

7.5.2.1 Notes

- Unlike **B$ReadRequest**, this call returns as soon the BIS Request Handler accepts the output file. This function call does not block waiting for a response from BIS.
- After writing a response, the service program will normally either call **B$ReadRequest** or terminate.
- The BIS Service Engine starts the service timer when this function returns. The program has 30 seconds to perform one of these actions:
  - Call **B$ReadRequest**.
  - Call **B$Exchange**.
  - Call **B$SetServiceTimeout**. A parameter of **0** will restart the service timer.
  - Terminate.

  If the service program processes for more than 30 seconds without performing one of the above functions, the BIS Service Engine assumes the service program is lost and begins termination processing (as if a **StopService** tag had been rendered).
- Other codes may also be returned, but that normally indicates a serious problem has occurred.

7.5.3 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 65 for a table of result codes and their interpretation.

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

7.5.4.1 Notes

- The default inactivity timeout period is **600** seconds (10 minutes). The section entitled "Session Inactivity Timeout" on page 28 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/IIS defers processing of this function until an **XMLExchange** tag is processed. The main implication of this restriction is that if the client starts the program and then browses pages that do not include an **XMLExchange** tag while the program calls **B$SetInactivityTimeout()** followed by **B$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** tag is not imminent.

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

7.5.5.1 Notes

- The default service timeout period is **30** seconds. The section entitled "Service Timeouts" on page 29 describes how the default may be changed for all BIS services on this server.
- The service timeout may also be set in a **.srf** file with the **SessionParms** tag.
- All calls to this function will restart the timer. Specify **0** to restart the timer without changing the value currently in effect.
- BIS/IIS defers processing of this function until an **XMLExchange** tag is processed. The main implication of this restriction is that if the client starts the program and then browses pages that do not include an **XMLExchange** tag while the program calls **B$SetServiceTimeout()** followed by **B$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** tag 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.
APP_POOL_ID	IIS	Returns the name of the application pool that is running in the IIS worker process that is handling the request.
APPL_MD_PATH	IIS	Retrieves the metabase path for the **Application** for the BIS server
APPL_PHYSICAL_PATH	IIS	Retrieves the physical path corresponding to the metabase path. IIS converts the **APPL_MD_PATH** to the physical (directory) path to return this value.
AUTH_PASSWORD	IIS	The value entered in the client's authentication dialog box. This variable is available only if Basic authentication is used.
AUTH_TYPE	IIS	The authentication method that the server uses to validate users when they attempt to access a protected script.
AUTH_USER	IIS	The name of the user as it is derived from the authorization header sent by the client, before the user name is mapped to a Windows account. This variable is no different from **REMOTE_USER**. If you have an authentication filter installed on your web server that maps incoming users to accounts, use **LOGON_USER** to view the mapped user name.
CERT_COOKIE	IIS	Unique ID for the client certificate, returned as a string. This can be used as a signature for the whole client certificate.

Variable	Platform	Description
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.
CERT_SECRETKEYSIZE	IIS	Number of bits in server certificate private key. For example, 1024.
CERT_SERIALNUMBER	IIS	Serial number field of the client certificate.
CERT_SERVER_ISSUER	IIS	Issuer field of the server certificate.
CERT_SERVER_SUBJECT	IIS	Subject field of the server certificate.
CERT_SUBJECT	IIS	Subject field of the client certificate.
CONTENT_LENGTH	All	The length of the content as given by the client.
CONTENT_TYPE	All	The data type of the content. Used with queries that have attached information, such as the HTTP queries **GET**, **POST**, and **PUT**.
DOCUMENT_ROOT	Apache	Contains the local directory from which the server is serving pages.
GATEWAY_INTERFACE	All	The revision of the CGI specification used by the server. The format is **CGI/*revision***. Example: **CGI/1.1**.
HTTP_*HeaderName*	All	The value stored in the HTTP header ***HeaderName***. Any header other than those listed below must be preceded by "**HTTP_**" in order for the **Value(*variable*, Server)** collection to retrieve its value. This is useful for retrieving custom headers. The server interprets any underscore (_) characters in ***HeaderName*** as dashes in the actual header. For example, if you specify **HTTP_MY_HEADER**, the server searches for a request header named **MY-HEADER**.
HTTP_ACCEPT	All	Returns the value of the **Accept** header. For example, "**image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel**".

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

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

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

Variable	Platform	Description
REMOTE_USER	IIS	The name of the user as it is derived from the authorization header sent by the client, before the user name is mapped to a Windows account. If you have an authentication filter installed on your web server that maps incoming users to accounts, use **LOGON_USER** to retrieve the mapped user name.
REQUEST_METHOD	All	The method used to make the request. For HTTP, this can be **GET**, **HEAD**, **POST**, and so on.
REQUEST_URI	Apache	The complete URI of the request.
SCRIPT_FILENAME	Apache	The complete file name of the script being executed.
SCRIPT_NAME	All	A virtual path to the script being executed. This is used for self-referencing URLs.
SERVER_ADDR	Apache	The IP address to which the request was sent.
SERVER_ADMIN	Apache	Contains the email address of the server's system administrator. (This is contents of the ServerAdmin configuration record.)
SERVER_NAME	All	The server's host name, DNS alias, or IP address as it would appear in self-referencing URLs.
SERVER_PORT	All	The server port number to which the request was sent.
SERVER_PORT_SECURE	IIS	A string that contains either **0** or **1**. If the request is being handled on the secure port, then this is **1**. Otherwise, it is **0**.
SERVER_PROTOCOL	All	The name and revision of the request information protocol. The format is ***protocol/revision***. Example: **HTTP/1.1**.
SERVER_SIGNATURE	Apache	The name and version of the Apache web server, plus the network name and port number on which the web server is running. Example: **Apache/2.0.55 (Unix) mod_ssl/2.0.55 OpenSSL/0.9.8a Server at arokh.liant.com Port 80</address>**
SERVER_SOFTWARE	All	The name and version of the server software that answers the request and runs the gateway. The format is ***name/version***. Example: **Microsoft-IIS/5.0**
SSL_CIPHER	Apache HTTPS	The name of the SSL cipher in use. Example: **RC4-MD5**
SSL_CIPHER_EXPORT	Apache HTTPS	Contains **true** if the cipher is an export cipher and **false** otherwise.

Variable	Platform	Description
SSL_CIPHER_ALGKEYSIZE	Apache HTTPS	The maximum number of bits permitted in the cipher's. Example: 128
SSL_CIPHER_USEKEYSIZE	Apache HTTPS	The number of bits actually in use in the cipher. Example: 128
SSL_CLIENT_A_KEY	Apache HTTPS	The signature algorithm used in the client key. Example: rsaEncryption
SSL_CLIENT_A_SIG	Apache HTTPS	The signature algorithm used in the client certificate. Example: sha1WithRSAEncryption
SSL_CLIENT_I_DN	Apache HTTPS	The client certificate issuer distinguish name subject. Example: /CN=neo
SSL_CLIENT_I_DN_CN	Apache HTTPS	The computer name of the client certificate issuer distinguish name subject. Example: neo
SSL_CLIENT_M_VERSION	Apache HTTPS	The client certificate's version. Example: 3
SSL_CLIENT_M_SERIAL	Apache HTTPS	The client certificate's serial number. Example: 1DFD4318000000000015
SSL_CLIENT_S_DN	Apache HTTPS	The client certificate distinguished name subject. Example: /C=US/ST=TX/L=Austin/O=Liant/OU=R&D/CN=Mike Schultz/emailAddress=m.schultz@liant.com
SSL_CLIENT_S_DN_C	Apache HTTPS	The country of the client certificate distinguished name subject. Example: US
SSL_CLIENT_S_DN_CN	Apache HTTPS	The contact of the client certificate distinguished name subject. Example: Mike Schultz
SSL_CLIENT_S_DN_Email	Apache HTTPS	The email address of the client certificate distinguished name subject. Example: m.schultz@liant.com
SSL_CLIENT_S_DN_L	Apache HTTPS	The location of the client certificate distinguished name subject. Example: Austin
SSL_CLIENT_S_DN_O	Apache HTTPS	The organization of the client certificate distinguished name subject. Example: Liant
SSL_CLIENT_S_DN_OU	Apache HTTPS	The organization unit of the client certificate distinguished name subject. Example: R&D
SSL_CLIENT_S_DN_ST	Apache HTTPS	The state of the client certificate distinguished name subject. Example: TX

Variable	Platform	Description
SSL_CLIENT_VERIFY	Apache HTTPS	Contains SUCCESS if the client verification was successful.
SSL_CLIENT_V_END	Apache HTTPS	The client certificate's validity end time. Example: Dec 16 20:27:44 2006 GMT
SSL_CLIENT_V_START	Apache HTTPS	The client certificate's validity end time. Example: Dec 16 20:17:44 2005 GMT
SSL_PROTOCOL	Apache HTTPS	The version of the SSL protocol. Example: SSLv3
SSL_SERVER_M_VERSION	Apache HTTPS	The server's certificate's version. Example: 1
SSL_SERVER_M_SERIAL	Apache HTTPS	The server's certificate's serial number. Example: 00
SSL_SERVER_S_DN	Apache HTTPS	The server certificate distinguished name subject. Example: /C=US/ST=Texas/L=Austin/O=Liant Software Corporation/OU=Development/CN=arokh.liant.com /emailAddress=m.schultz@liant.com
SSL_SERVER_S_DN_C	Apache HTTPS	The country of the server certificate distinguished name subject. Example: US
SSL_SERVER_S_DN_CN	Apache HTTPS	The computer name of the server certificate distinguished name subject. Example: arokh.liant.com
SSL_SERVER_S_DN_Email	Apache HTTPS	The email address of the server certificate distinguished name subject. Example: m.schultz@liant.com
SSL_SERVER_S_DN_L	Apache HTTPS	The location of the server certificate distinguished name subject. Example: Austin
SSL_SERVER_S_DN_ST	Apache HTTPS	The state of the server certificate distinguished name subject. Example: Texas
SSL_SERVER_S_DN_O	Apache HTTPS	The organization of the server certificate distinguished name subject. Example: Liant Software Corporation
SSL_SERVER_S_DN_OU	Apache HTTPS	The organization unit of the server certificate distinguished name subject. Example: Development
SSL_SERVER_I_DN	Apache HTTPS	The server certificate issuer's distinguished name subject. Example: /C=US/ST=Texas/L=Austin/O=Liant Software Corporation/OU=Development/CN=arokh.liant.com /emailAddress=m.schultz@liant.com

Variable	Platform	Description
SSL_SERVER_I_DN_C	Apache HTTPS	The country of the server certificate issuer's distinguished name subject. Example: **US**
SSL_SERVER_I_DN_CN	Apache HTTPS	The computer name of the server certificate issuer's distinguished name subject. Example: **arokh.liant.com**
SSL_SERVER_I_DN_Email	Apache HTTPS	The email address of the server certificate issuer's distinguished name subject. Example: **m.schultz@liant.com**
SSL_SERVER_I_DN_L	Apache HTTPS	The location of the server certificate issuer's distinguished name subject. Example: **Austin**
SSL_SERVER_I_DN_O	Apache HTTPS	The organization of the server certificate issuer's distinguished name subject. Example: **Liant Software Corporation**
SSL_SERVER_I_DN_OU	Apache HTTPS	The organization unit of the server certificate issuer's distinguished name subject. Example: **Development**
SSL_SERVER_I_DN_ST	Apache HTTPS	The state of the server certificate issuer's distinguished name subject. Example: **Texas**
SSL_SERVER_A_KEY	Apache HTTPS	The signature algorithm of the server's key. Example: **rsaEncryption**
SSL_SERVER_A_SIG	Apache HTTPS	The signature algorithm of the server's certificate. Example: **md5WithRSAEncryption**
SSL_SERVER_V_END	Apache HTTPS	The server certificate's validity end time. Example: **Jan 13 08:13:27 2006 GMT**
SSL_SERVER_V_START	Apache HTTPS	The server certificate's validity start time. Example: **Dec 14 08:13:27 2005 GMT**
SSL_VERSION_INTERFACE	Apache HTTPS	The version of the SSL interface. Example: **mod_ssl/2.0.55**
SSL_VERSION_LIBRARY	Apache HTTPS	The version of the SSL library. Example: **OpenSSL/0.9.8a**
UNMAPPED_REMOTE_USER	IIS	The name of the user as it is derived from the authorization header sent by the client, before the user name is mapped to a Windows account. This variable is no different from REMOTE_USER. If you have an authentication filter installed on your web server that maps incoming users to accounts, use LOGON_USER to view the mapped user name.
URL	IIS	Gives the base portion of the URL.

Appendix B. XML Exchange Request File Format

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

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

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

< bis:content > ***payload data*** **< / bis:content >**	Contains the content part of the request (such as form variables POSTed back to the server). This element will be empty if there is no content data in the request—as is typically true of the first (GET) request.
< bis:cookies > **< bis:cookie name=*name* >** ***cookie data*** **< /bis:cookie >** **...** **< / bis:cookies >**	Contains an attributed **<bis:cookie>** element for each cookie that was transmitted with the request.
< bis:query-params > **< bis:query-param name=*name* >** ***parameter data*** **< / bis:query-param >** **...** **< / bis:query-params >**	Contains an attributed **<bis:query-param>** element for each query parameter that was transmitted with the request.
< bis:server-variables > **< bis:server-variable name=*name* >** ***server variable data*** **< / bis:server-variable >** **...** **< / bis:server-variables >**	Contains an attributed **<bis:server-variable>** element for each server variable associated with this request.

The content of a sample request file is below. Note that this is also visible in the trace output, if tracing is enabled. Also note that the **<bis:content>** section is application-dependent. This particular example is from the http://localhost/xbis10/samples/sample1 application with the following data entered into the form fields:

Containing Element	Element	Attribute	Value
bis:content	firstname.Q12		John
	lastname.Q20		Smith
	email.Q33		J.Smith@xcentrisity.org
	gender.Q52		Male
bis:cookies	cookie	Name="BISKIT"	VH9rQjrTe1b54iQ2vnTkS898YaF7
bis:query-params	query-param	Name="__xmlexch"	HQaD

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 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. Note that the XML tags are indented to make the example easier to read.

```
<?xml version="1.0" encoding="UTF-8" ?>
<bis:request xmlns:bis="http://www.xcentrisity.com/2003/bis/request">
<bis:content>  <email.Q33__1>
<![CDATA[
J.Smith@xcentrisity.org
]]>
</email.Q33__1>
<firstname.Q12>
<![CDATA[
John
]]>
</firstname.Q12>
<gender.Q52>
<![CDATA[
Male
]]>
</gender.Q52>
<lastname.Q20>
<![CDATA[
Smith
]]>
</lastname.Q20>
</bis:content>
<bis:cookies>
<bis:cookie name="BISKIT">[1]
<![CDATA[
pR+APfDpGuK1+ZuSMAD5zLas+MZ3
]]>
</bis:cookie>
<bis:cookie name="BISKIT">
<![CDATA[
VH9rQjrTe1b54iQ2vnTkS898YaF7
]]>
</bis:cookie>
</bis:cookies>
<bis:query-params>
<bis:query-param name="__xmlexch">
<![CDATA[
HQaD
]]>
</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/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-
powerpoint, application/msword, */*
HTTP_ACCEPT_LANGUAGE:en-us
HTTP_CONNECTION:Keep-Alive
HTTP_HOST:localhost
HTTP_REFERER:http://localhost/xbis10/samples/sample1/default.srf?trace=page,ex
chfiles,file
HTTP_USER_AGENT:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET
CLR 1.1.4322; InfoPath.1; .NET CLR 2.0.50727)
```

[1] The **BISKIT** cookie appears twice because a new session for the **SAMPLE1** program was created when this sample was run from the main menu. The session created for the main menu continues to run, as it is isolated from the **SAMPLE1** session.

```
HTTP_COOKIE:BISKIT=VH9rQjrTe1b54iQ2vnTkS898YaF7;
BISKIT=pR+APfDpGuK1+ZuSMAD5zLas+MZ3
HTTP_CONTENT_LENGTH:90
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/x-
shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*
Accept-Language: en-us
Connection: Keep-Alive
Host: localhost
Referer:
http://localhost/xbis10/samples/sample1/default.srf?trace=page,exchfiles,file
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR
1.1.4322; InfoPath.1; .NET CLR 2.0.50727)
Cookie: BISKIT=VH9rQjrTe1b54iQ2vnTkS898YaF7;
BISKIT=pR+APfDpGuK1+ZuSMAD5zLas+MZ3
Content-Length: 90
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/Xbis10
  ]]>
  </bis:server-variable>
  <bis:server-variable name="APPL_PHYSICAL_PATH">
  <![CDATA[
d:\dev\BIS10.00\rmc85\bis\
  ]]>
  </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[
90
  ]]>
  </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/x-shockwave-
flash, application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*
  ]]>
  </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[
90
  ]]>
  </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=VH9rQjrTe1b54iQ2vnTkS898YaF7; BISKIT=pR+APfDpGuK1+ZuSMAD5zLas+MZ3
  ]]>
  </bis:server-variable>
  <bis:server-variable name="HTTP_HOST">
  <![CDATA[
localhost
  ]]>
  </bis:server-variable>
  <bis:server-variable name="HTTP_REFERER">
  <![CDATA[
http://localhost/xbis10/samples/sample1/default.srf?trace=page,exchfiles,file
  ]]>
  </bis:server-variable>
  <bis:server-variable name="HTTP_URL">
  <![CDATA[
```

```
/xbis10/samples/sample1/default.srf?__xmlexch=HQaD
  ]]>
  </bis:server-variable>
  <bis:server-variable name="HTTP_USER_AGENT">
  <![CDATA[
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322;
InfoPath.1; .NET CLR 2.0.50727)
  ]]>
  </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="PATH_INFO">
  <![CDATA[
/xbis10/samples/sample1/default.srf
  ]]>
  </bis:server-variable>
  <bis:server-variable name="PATH_TRANSLATED">
  <![CDATA[
d:\dev\BIS10.00\rmc85\bis\samples\sample1\default.srf
  ]]>
  </bis:server-variable>
  <bis:server-variable name="QUERY_STRING">
  <![CDATA[
__xmlexch=HQaD
  ]]>
  </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_PORT">
  <![CDATA[
1908
  ]]>
  </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[
/xbis10/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="UNMAPPED_REMOTE_USER" />
  <bis:server-variable name="URL">
  <![CDATA[
/xbis10/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. UNIX BIS 8 Compatibility Issues

There are few differences between UNIX BIS 8 and BIS/Apache 10 to note.

D.1 Apache Configuration

BIS 8's Apache configuration was accomplished using environment variables. In BIS 10, this configuration is now accomplished using Apache configuration directives. Where possible, the name of the configuration directive is similar to the environment variable used by BIS 8. The table below gives the BIS 8 Apache environment variable name, the equivalent BIS 10 configuration directive and the location of the directive. **mod_xbis.conf** refers to the Request Handler configuration file in the Apache configuration directory. **xbis.conf** refers to the Service Engine's configuration file in the **/etc** directory.

BIS 8 Configuration Option	BIS 10 Configuration Option	Location
MASTER_DEBUG	BISMasterTrace	mod_xbis.conf
STENCIL_DEBUG	BISStencilDebug	mod_xbis.conf
MESSAGE_DEBUG	Not Supported	
MAIN_DEBUG	BISMainDebug	mod_xbis.conf
LOG_TRACE_FILES	Not Supported	
KEEP_TRACE_FILES	BISKeepTraceFiles	mod_xbis.conf
BIS_TRACE_SUFFIX	BISTraceSuffix	mod_xbis.conf
RUNBIS_TRACE_FILE	BISTraceFile	mod_xbis.conf
REFRESH_DIRECTORY	BISRefreshDirectory	mod_xbis.conf
BIS_SESSION_INACTIVITY_TIMEOUT	InactivityTimeOut	xbis.conf
BIS_SERVICE_TIMEOUT	ServiceTimeOut	xbis.conf
BISErrorTextConfig	BISErrorMessage	mod_xbis.conf
BISErrorHTML	Not Supported	

Appendix E. Regular Expression Syntax

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

E.1 Metacharacters

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

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

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`".

E.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

E.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, and so forth; or a Unicode character block, for example, IsBasicLatin.

Appendix F. 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. If a field is omitted, it is replaced by a dash (-) character.

F.1 Log File Location

Under IIS, the log files are 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, composed from the UTC time when the log file was created:

yyyyMMddhhmmss.log

On UNIX, the log files are written to the directory specified by the **LogDirectory** (or **LogDir**) directive in the **/etc/xbis.conf** file. This is usually a subdirectory of the configured "temp" directory, such as

/var/tmp/bislogs

Each log file has the following name, composed from the date portion of the UTC time when the log file was created:

yyyyMMdd.log

F.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 format can be sorted. 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**
Z	-	**licenseInfo**	**idBaseLicense, idSerialNumber, nUseCount, strLicenseKey**

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.

The record types are nested: there will normally be a matching **sessionEnd** record for each **sessionBegin** record. The **licenseInfo** record is an exception: this record will occur by itself. For BIS/IIS, the **licenseInfo** record is written after BIS is loaded and the first service is started. Because much of the BIS subsystem is unloaded after a period of idle time and automatically reloaded on demand, this record may occur more than once in a given log file.

F.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
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**

Record Type	Field	Description
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
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

Record Type	Field	Description
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
licenseInfo	idBaseLicense	The base license ID from the Service Engine license file.

Record Type	Field	Description
	idSerialNumber	The serial number from the Service Engine license file.
	nUseCount	The use count from the Service Engine license file.
	strLicenseKey	The contents of the **WebServer** license key.

Appendix G. 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/xbis10/verify/default.srf?trace=page
```

  Trace output will appear at the bottom of the page, and this will include the BIS Service Engine startup messages that should reveal the problem.

- **Symptom:** An error 500 occurs.

- **Possible Cause:** A replacement tag precedes the **Handler** tag.

- **Suggestion:** The only tags allowed before the **Handler** tag are comment tags. Move all tags that precede the **Handler** tag to follow it.

- **Symptom:** One of the following error messages is reported:

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

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

Could not write the trace file to the directory "D:\ Documents and
Settings\UserID\Local Settings \Temp\": the error code was 80004005.
```

- **Suggestion:** To correct this error, give the IWAM_* account write access to this directory. See the "Troubleshooting" appendix in the User's Guide for more information.

- **Symptom:** The following error message appears in the web browser:

```
Server Error

LoadLibrary failed.
```

- **Possible Cause:** The **Handler** tag is missing, invalid, or refers to a missing or invalid library.

- **Suggestion:** Make sure that your **.srf** page has a **Handler *}}** tag, and that this tag is the first non-comment/non-include tag in the file. For Windows, it must also appear in the first 4096 characters of the **.srf** file.

Appendix H. Configurating BIS/IIS after Installation

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

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

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

H.1 Command Line Configuration

BIS is self-registering. Registration is performed by the **XBIS.EXE** program, which can be found in the installation directory (normally **C:\Program Files\Liant\BIS10**). Registration includes these three steps:

- The BIS Service Engine contained in **XBIS.EXE** is registered.
- The runtime system contained in **RMCBL10R.DLL** is registered.
- The *Run As* identity, that is, the identity that will be used to execute service programs, is set.

The server registration syntax is:

```
XBIS registration-options
```

The registration options are detailed below:

/REGSERVER Registers the Service Engine. Also registers the runtime system located in the same directory as **XBIS.EXE**.

/UNREGSERVER Unregisters the BIS Service Engine and the runtime system.

/SHOWSERVER Displays a dialog box that shows the location of the currently registered BIS and Service Engine.

The server registration option has three additional variations:

/REGSERVERQ Quietly registers the BIS Service Engine and the runtime system located in the same directory as **XBIS.EXE**. No confirmation dialog box is displayed.

/REGSERVERO Only registers the BIS Service Engine. The runtime system registration remains unchanged. This is useful if you want to install the runtime system in a directory separate from the BIS Service Engine.

/REGSERVERQO Combines the above two options.

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

```
1. /REGSERVER:pathname
2. /REGSERVERQ:pathname
```

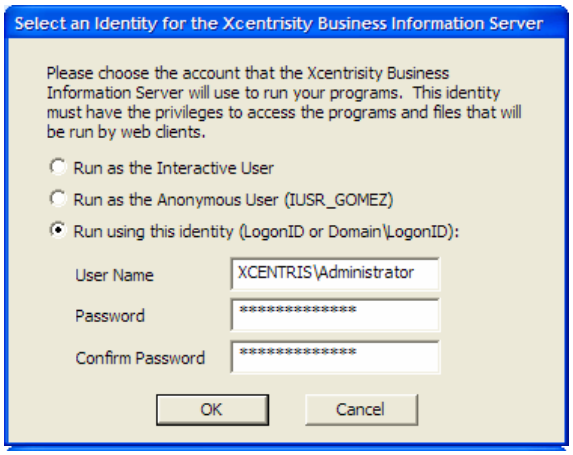
```
3. /REGSERVER:directory
4. /REGSERVERQ:directory
```

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

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

H.1.1 Configuring the Run As Logon ID

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



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

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

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

/RUNASI
/RUNASIP

Causes the server to run as the "**INTERACTIVE USER**". This is the identity of the user that is logged on to the server's console. This is most useful for developers but is not recommended for deployment.

If the **P** suffix is specified, BIS prompts if an error occurs.

/RUNASL
/RUNASLP

Runs the server under the identity of the launching (usually anonymous) user. This will normally be the account named **IUSR_*machinename***, where ***machinename*** is the name assigned to the machine.

For example, if your machine is named **HILO**, the anonymous user's name is **IUSR_HILO**. It is possible for a system administrator to change this, either for all IIS accounts or for just the BIS. If the name of the machine was changed after IIS was installed, this will be the original name of the machine, not the current name. In this case, please see "Manual Configuration", below.

Note that this account usually has very limited privileges and BIS will not even be able to start unless you manually give this account write permission in the BIS installation directory. BIS will not be able to access files in other directories, unless you also give it access to those directories, and will not be able to access files on any network volumes unless your machine is joined to a domain and this name is known to the domain server. See your system administrator for details.

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

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

Runs the server using the specified identity. This is the recommended option. ***id*** is the login ID and ***pw*** is the password. The password is encrypted by Windows, is stored in the registry, and is not retrievable as plain text once the server is registered. However, caution is required when embedding a clear-text password in a batch file that issues the **/RUNAS** command.

If an ***id*** is specified without a ***pw***, the program prompts for the password. This may be a good compromise between convenience and security.

Either the ***id*** or the ***pw*** may be quoted with single or double quotes (required if either contains spaces). The entire parameter string may also be quoted.

Examples:

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

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

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

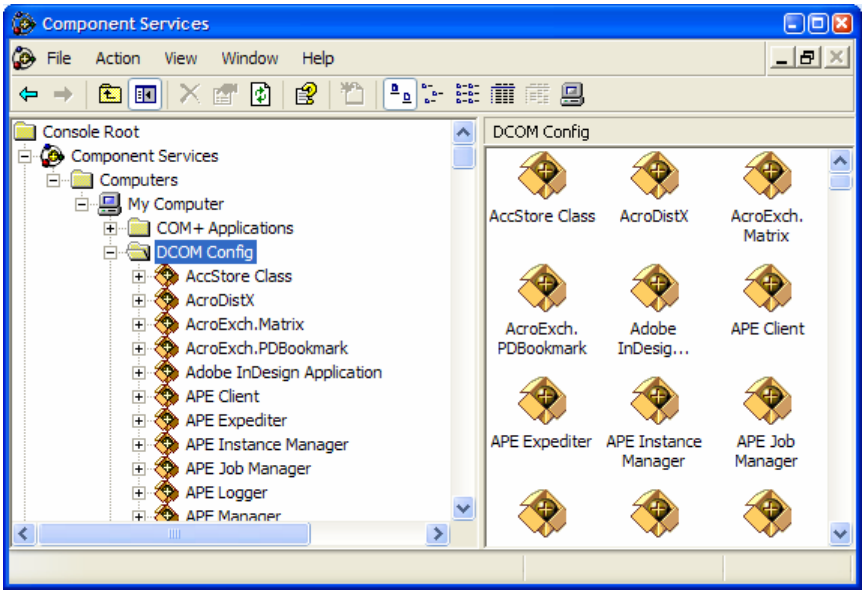
H.1.2 Retrieving or Changing the Configured Identity

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

There are two ways to start the utility:

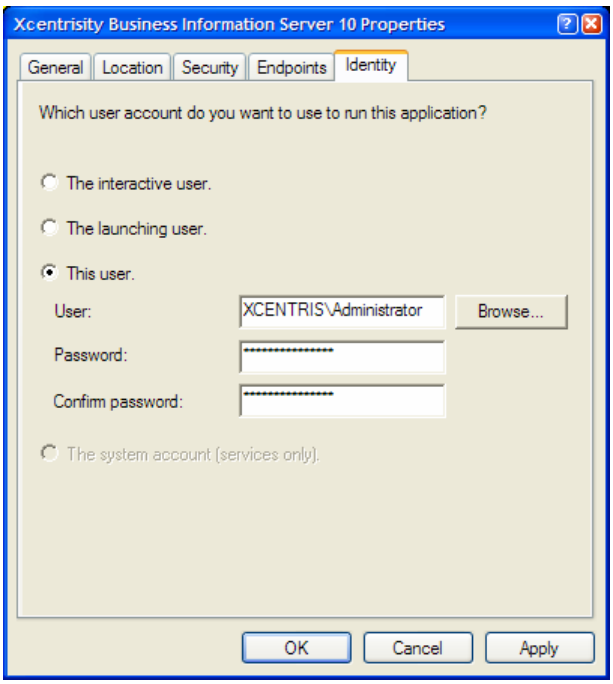
- Select Start → Control Panel → Adminstrative Tools → Component Services. (Alternatively, select Start → Run, enter dcomcnfg in the "Open" box, and click the OK button.)

- Select Start → Control Panel → Adminstrative Tools → Component Services. The program should look like this:



3. Find Xcentrisity Business Information Server 10 in the list, right-click, and select Properties from the popup menu.

4. Click on the Identity tab. The dialog box depicted below displays the current *Run As* configuration.



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

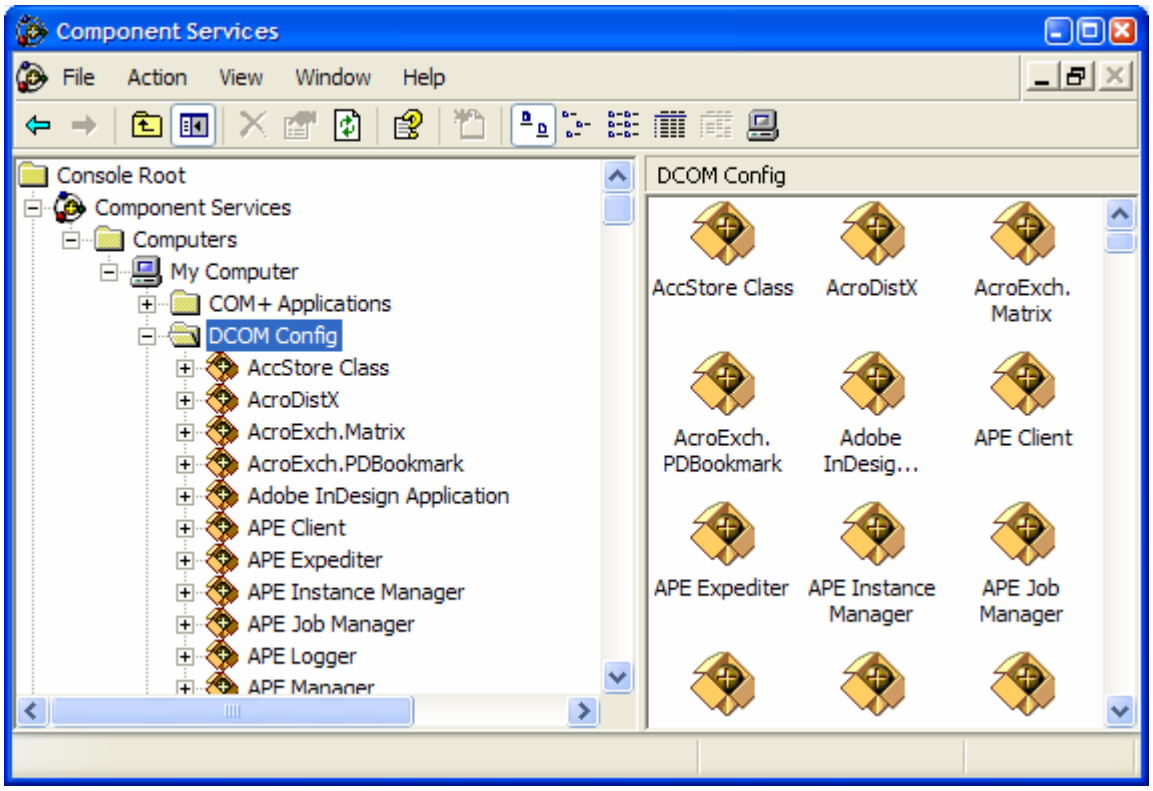
H.2 Manual Configuration

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. The program should look like this:



3. Locate Xcentrisity Business Information Server 10 in the list, right-click, and select Properties from the popup menu.

4. Click the Identity tab, then This user. Enter the user ID and the password that you want to use to run service programs under Business Information Server. Then click the Apply button.

5. Click the Security tab and under Launch Permissions, click Customize and then click Edit. Click Add and enter the name of your anonymous internet account (see below). Click the Add button; make sure Allow is checked next to Launch Permission and click OK. Then click Apply.

6. Still on the Security tab, repeat the above step for Access Permissions.

7. You do not need to change Configuration Permissions. Click OK to close the dialog box.

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

To determine the name of your anonymous internet account:

1. Select Start → Control Panel → 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 BIS service program. This will be XBIS10 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 web site.

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

H.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 box.

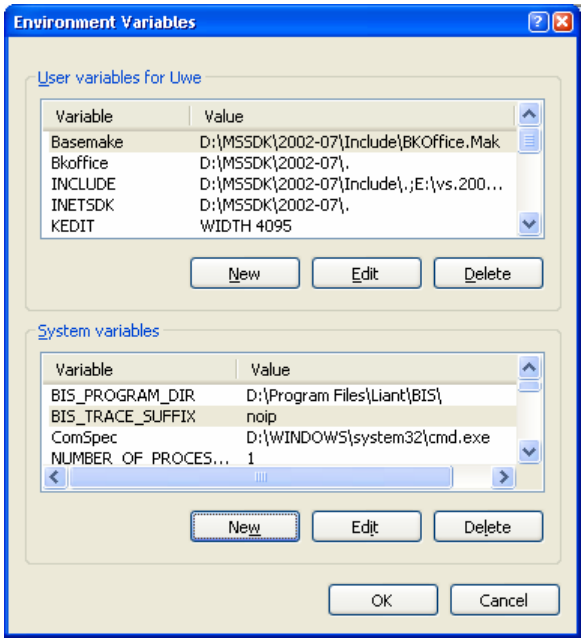


Figure 7-2. The Environment Variables Dialog

The changes take effect immediately.

H.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.

H.5 Notes

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

Appendix I. Configuration after Installation (UNIX/Apache)

I.1 Configuring Apache

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

```
Include conf/mod_xbis.conf
```

Or, if available, the **mod_xbis.conf** may be placed in the **/etc/httpd/conf.d** directory, which will circumvent the necessity of editing the main **httpd.conf** configuration file.

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

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

```
LoadModule xbis_module  modules/mod_xbis2.so
AddHandler bis-stencil  srf
AddType    text/html    srf
AddType    text/x-component .htc
```

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

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

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

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

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

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

The **BISTraceDirectory** directive indicates the directory where trace files are to be written. The default for this directive is **/tmp**. If this directive does not specify an absolute path, it is assumed to be relative to **/tmp**.

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

The **BISKeepTraceFiles** directive controls if trace files are to be kept after a session completes. The value of **Off** is the default, and it will cause trace files to be deleted, unless a FILE Trace tag option requests that they be kept. The value of **On** will cause trace files to be retained regardless of the presence of a FILE trace option.

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

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

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

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

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

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

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

The **BISRefreshDirectory** directive names a directory where server responses are stored temporarily that may be needed if the client agents (web browsers) request a refresh. (See the XMLExchange tag.) 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, the BIS Request Handler will not attempt to provide correct responses to refresh requests which will lead to unnecessary session sequence errors.

```
BISErrorMessage ErrorName Error Text
```

The **BISErrorMessage** directive allows the text for the BIS Request Handler's error messages to be overridden, changing it to support a language besides English. The first operand of the directive is the name of the error to be overridden. The remainder of the directive is the new text to be displayed when **ErrorName** is encountered. The current set of the Request handler's error names and their text are present within **mod_xbis.conf** as commented out BISErrorMessage directives.

```
BISSesDaemonKey xxxxxxxx
```

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

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

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

```
<Directory Directory-Path>
    SetEnv BIS_ROOT_PATH /xbis10/samples
    DirectoryIndex default.srf
</Directory>
```

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

The **SetEnv** directive demonstrates setting a server environment variable to be passed to the service program.

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

I.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 a configuration file, its command line and its environment. The configuration file is usually named **/etc/xbis.conf**, but this can be changed by the **-f** command-line option. Each line in the configuration file is either a blank line, comment line or an option line. A comment line is a line in which the first nonblank character is a "**#**" character. On an option line,

the line begins with a keyword, which is followed by one or more spaces or tabs and then by the option value. A "**#**" character may follow the option value to introduce an in-line comment.

The configuration file option keywords are:

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

Options on the Service Engine daemon's command line may modify the configuration as determined by the configuration file and the built-in defaults. On systems other than AIX, the command-line options are in a string that is assigned to an environment variable named **OPTIONS**. All of the Service Engine's environment variables, including **OPTIONS**, are set in a file named **/etc/sysconfig/xbis**. This file is created during the install of BIS.

On AIX systems, the Service Engine runs as a SRC subsystem. The command-line options are stored in an ODM object created by the **mkssys** command during the install of BIS. Additionally, the **startsrc** command which starts the Service Engine may specify command-line options and environment variables. A **startsrc** command is inserted into the **/etc/inittab** file during installation.

The command-line options are:

-f *file*	Specifies the name of the Service Engine configuration file. If this option is present, it must be the first option on the command line. If omitted, the configuration file name defaults to **/etc/xbis.conf**.
-c *count*	Specifies the maximum number of service (child) processes. This is normally set to **9999** to indicate that the number of service processes is limited only by the license, but it may be set to a smaller value as a "throttle."
-i *count*	Specifies the number of idle service (child) processes. This is normally set to **1** but a small increase in this may improve response time on a server which receives many requests in rapid succession.
-T *timeout*	Default service timeout, in seconds. This is the preferred way to set the default service timeout. If **BIS_SERVICE_TIMEOUT** is set in the Apache configuration file for BIS (**bis.conf**), the Request Handler uses that value to override the value of the **-T** option. Doing so delays the start of each service program slightly.
-u *user*	Specifies the UNIX user name used by each service (child) processes. Although the Service Engine daemon process runs as **root**, each of the child service processes runs as the user specified by this option. This determines the files that a service process can read and write, as well as the home directory of each service process.
-t *dir*	Specifies the name of the directory where temporary files are created.
-r	If specified, copies of all request and response files are saved in the temporary directory. This is a debugging tool, typically used during development of a web site.
-A	If specified, the name of the temporary file is passed to the service program in the LINKAGE SECTION, just as if the A parameter were included in 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/xbis.conf**) may be edited or (on systems other than AIX) the file **/etc/sysconfig/xbis** may be edited. If the configuration file is changed, the Service Engine can be instructed to reread it by using a **kill** command to send the Service Engine daemon a **SIGHUP** signal. On AIX, a **refresh** command may be used for the same purpose. However, the Service Engine does not read **/etc/sysconfig/xbis** directly. Instead, the shell script which starts the Service Engine reads this file. For any changes to take effect, the Service Engine must be restarted, either by restarting the operating system, by changing the runlevel, or by executing the shell script which starts the Service Engine (**/etc/init.d/xbisengd**). This script accepts one parameter, which must be one of the following:

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

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

I.3 xbisctl Utility

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

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

start	Starts the Service Engine and the Session/Logging daemon.
stop	Stops the Service Engine and the Session/Logging daemon.
status	Displays a one-line status for Service Engine and the Session/Logging daemon.
refresh	Refreshes the Service Engine and the Session/Logging daemon. This tells the BIS daemons to reread their configuration file.
sessions	List the current sessions.
kill	Terminate a session.
exit	Stop reading standard input. Alternatively, press ctrl-D to end input.

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

I.4 SRC Commands

On AIX, the Service Engine and the BIS Session/Logging daemon run as SRC subsystems. This table summarizes the various SRC commands. The Service Engine is a subsystem named "**xbisengd**." The Session/Logging daemon is a subsystem named "**xbissesd**." The pair of them are usually started and stopped together, so they defined as a subsystem group named "**xbis**."

On the following commands, the **-s** parameter specifies the subsystem name (**xbisengd** or **xbissesd**). The **-g** parameter specifies the subsystem group (**xbis**).

startsrc	Starts a subsystem.
stopsrc	Stops a subsystem. The **-f** parameter causes a forced (quick) stop.
refresh	Refreshes a subsystem. This tells the BIS subsystem to reread its configuration file. It is usually best to use the **-g** parameter and specify the xbis group.
lssrc	List the status of a subsystem. The **-l** parameter requests a long (detailed) status. The **-a** parameter specifies all subsystems.

Appendix J. Creating a BIS/IIS Virtual Directory

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

C:\Program Files\Liant\BIS10\SupportTools

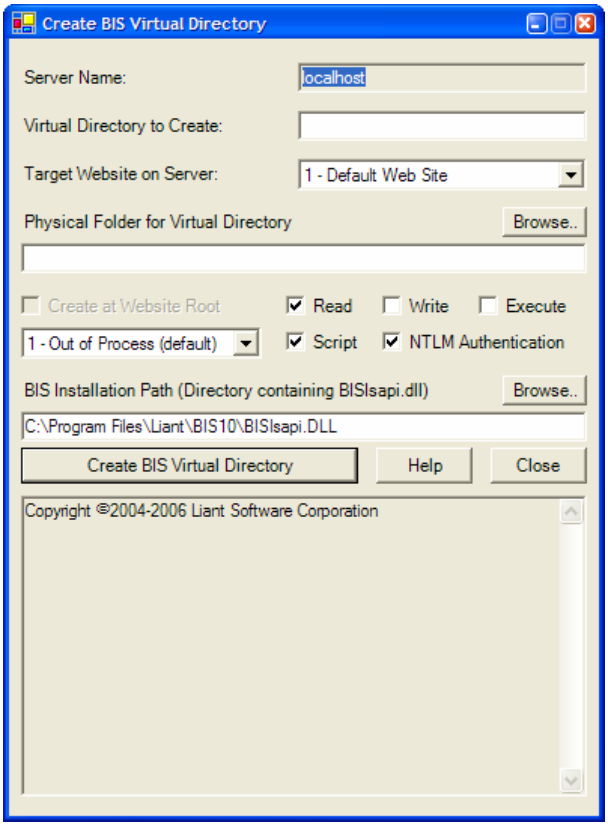
This program can also be downloaded from the Liant support web site or obtained from Liant support.

J.1 Running the BISMkDir Program

To launch this program, start Windows Explorer, navigate to the above directory and double-click on the **BISMkDir** icon. You can also follow these steps:

Start → Run → C:\Program Files\Liant\BIS10\BISMkDir.exe

When execution begins, you will see the dialog box depicted to the right. This dialog box has the following fields:



Server Name

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

Virtual Root Name

Enter the name of the virtual directory that you wish to created. For example, the default installation creates a virtual directory named **XBIS10**.

Physical Folder for Virtual Root

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

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

http://localhost/xbis10/samples/default.srf

The requested content is served from:

C:\inetpub\wwwroot\xbis10\samples\default.srf

This is because the BIS installer creates a physical directory named **XBIS10** in the default web tree, and copies the sample programs into this directory. The installer then creates a virtual root directory named **XBIS10**, configures it so it runs a BIS application (see below) and points it at the previously created physical directory.

Notes:

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

Site #

If your IIS server handles more than one web site, enter the number of the web site that will contain this virtual directory. Web site #1 is the default web site, and is the only web site that can be served by Windows2000 and Windows XP Professional.

Checkboxes

The checkboxes control how the virtual directory is created.

- **Root Dir** is reserved for future use.
- **Read** determines if web clients will have read permission to this virtual directory. This must be checked if BIS programs will be run in this directory.
- **Write** determines if web clients will be able to write to this virtual directory. **Note:** This should never be enabled, as it is a security risk.
- **Execute** determines if programs can be executed in this virtual directory. This should not be enabled unless you are also using this directory as a CGI-type directory and plan to run programs out of this virtual directory on the web server.
- **Script** determines if scripts can be executed in this virtual directory. This must be checked f if BIS programs will be run in this directory.
- **NTLM Authentication** should be checked to use this kind of authentication in this directory. In general, this box should be checked.

BIS Installation Path

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

J.2 Creating the Directory

When all of the above fields are filled, press the **Create VDir** button to begin the process of creating the virtual directory. Please be patient—it can take 30 seconds to create the directory. Once the program

finishes, messages will appear in the box at the bottom of the window. At that point, you can create another directory or close the program.

J.3 Testing the New Directory

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

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

(The above text can be copied from http://xcentris.org/biskit/sampletext.htm and pasted into your page).

Then enter the following into your web browser:

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

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

You should see a page containing only this text:

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

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

Appendix K. 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 L. 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 BIS/IIS):

```
build.bat
```

or (for BIS/Apache):

```
build.sh
```

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

Appendix M. Glossary

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

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

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

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

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

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

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

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

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

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

Session. A “stateful” sequence of HTTP request/response interactions between a web user agent (for example, browser) and a BIS Request Handler. The session identification is preserved in the user agent by means of a session cookie provided in the response to the first request of the session. All subsequent requests containing that cookie are assumed to be for the designated session.

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

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

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

URL. A Uniform Resource Locator, the location of a resource on the internet. A URL is a type of URI (Uniform Resource Identifier), and consists of a *scheme* (in this context, HTTP or HTTPS), the name of a *machine* (sometimes also called the *authority*), and a *path* to a resource (for example, a file). For example, http://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.

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