Rational Build Forge General Maintenance

Understanding Maintenance needs for Production Environments

William Frontiero & Joseph Bucanelli

August 18, 2011

1 Introduction

This whitepaper is designed as a supplement to the Build Forge Online Help Documentation. This document should not be seen as a replacement for the Online Help Manual. Build Forge Administrators and Release Engineering teams should review this document before making any changes to a new or existing environment. The information supplied is not based on IBM recommendations.
This white paper is intended to assist in creating scalable and manageable Build Forge environments. The paper is broken down into individual components due to certain software requirements. Build Forge out of the box requires: Database backend, Java Application Server (Tomcat), Web Server (Apache with PHP), and Precompiled Perl Executables (Perl Engine). Each component has its own unique options when considering management and scalability.
After reviewing this whitepaper, apply changes to a test environment before releasing in production. Some of the topics discussed in this paper result in data removal and aggressive purging. Please note that these topics should be discussed prior to implementation.

2 Overview of the topic

Topic #1 Web Server (PHP / Apache / IHS) Management

- What files are crucial to the Build Forge Application
- What Directives are Required for Build Forge Operation
- What Log files contain Build Forge specific information
- Manually stopping and starting the Web Server Component

Topic #2 Application Server (Tomcat / WAS) Management

- Install / Configuration / Web Application / Logs
    - WebSphere
        - Windows
        - Unix/Linux
    - Tomcat
        - Windows
        - Unix/Linux
- Manually Starting and Stopping the Application Server Component
    - WebSphere
        - Windows
        - Unix/Linux
    - Tomcat
        - Windows
        - Unix/Linux

Topic #3 Management Console Settings

- Manifest / Server Test / Max Processes / Max Jobs (How this impacts the system)
- Managing System Messages
- Managing Purge options (Project/Build Level)
- Managing Step log information to reduce Database size
- Know what is scheduled

Topic #4 Database / Engine Management

- Where are the Database Logs (What logs require Management)
- Where are the Engine Logs (When should they be archived)
- Where are the Build Forge Tables (What tables should we monitor)
- What is a Schema and why should we care about it
- How to ensure my Schema is up to date
- How to predict and monitor a healthy database growth rate

3 Topic components and definitions

Web Server #1 (HTTP Server for Web Interface)

HTTP Server (Apache or IHS) is necessary for hosting the Build Forge Management Console. By Default the Web Server will listen for HTTP traffic on Port 80. The Build Forge Management Console is designed to run on Apache Web Server (Apache) or IBM HTTP Server (IHS). The Web Server Component by default is installed to:

Windows: `<BF_HOME>\Apache\....`
Unix/Linux: `<BF_HOME>/server/apache/…`

Application Server #2 (Services Layer)

Java Application Server (Tomcat or WebSphere Application Server) is necessary to host the Java Services Layer. The Application Server is a Java based Server listening on both secure and non-secure API TCP ports (Default: Secure 49150, unsecured 3966) and UI TCP ports (Default: Secure 8443, unsecured 8080 for Tomcat, Default: Secure 9443, unsecured 9080 for WebSphere Application Server). Build Forge Installation by default will install Tomcat and deploy the Services Layer Web Application.

- Web Server Component default installation Path
  - Windows: `<BF_HOME>\Apache\tomcat\...`
  - Unix/Linux: `<BF_HOME>/server/tomcat/…`
- Services Layer Web Application default deployment Path
  - Windows: `<BF_HOME>\apache\tomcat\webapps\rbf-services\...`
  - Unix/Linux: `<BF_HOME>/server/tomcat/webapps/rbf-services/…`
- Services Layer default URL:
  - `https://<bf_Server_host_name>:8443/rbf-services`

Management Console #3 (MC)

The Server Side Web Application is written primarily in html/PHP/JavaScript. The MC is hosted by the Installed Web Server (Apache or IHS). The MC is used as a central point of Build Forge Administration and general usage. The Management Console was originally designed to make direct Database calls via PHP and Database vendor specific SQL drivers.

- MC URL structure: (Default Install)
  - `http(s)://<bf_Server_host_name>:<web_server_port>`
    - Web Server Port 80 for non SSL (unsecured)
    - Web Server Port 443 for SSL (Secure)
- MC Install location: (Default Install)
  - Windows: `<BF_HOME>\webroot\...`
  - Unix/Linux: `<BR_HOME>/webroot/…`

Database #4 (Data Warehouse)

Database as implemented by Build Forge: The Database is a central point of data warehousing for all Build Forge information and configuration needs. The Database must be one of 4 vendors, Oracle, DB2, MySql or Microsoft SQL Server. Review Build Forge 7.1.2.X Database Requirements.
Build Forge also requires the Vendor specific Client libraries for Perl and PHP database access and JDBC driver for the Java Services Layer.

Engine #5

The Build Forge Engine comprises of Perl and Java based components responsible for executing and automating tasks configured by the Management Console. The Engine is still primarily Perl based when executing Build Forge Jobs and interacting with Agents at runtime.

Here is a list of Engine Components found in the Build Forge Install Directory

- Windows <BF_HOME>*
  - bfmessages.exe
  - bfproject.exe
  - bfpwencrypt.exe
  - bfrefresh.exe
  - bfservertest.exe
  - bfstepcmd.exe
  - buildforge.exe
- Unix/Linux <BF_HOME>/Platform/*
  - bfmessages
  - bfproject
  - bfpwencrypt
  - bfrefresh
  - bfservertest
  - bfstepcmd
  - buildforge

4 Management Options (Per Component)

Web Server #1 (HTTP Server for Web Interface)

1.) What Files are Crucial to the Build Forge Application

To administer and manage the HTTP server, you must review the Server's Master Configuration File. This files purpose is to configure the Web Server on startup. (httpd.conf)

- File location in a Default installation of Apache
  - Windows: `<BF_HOME>\apache\conf\httpd.conf`
  - Unix/Linux: `<BF_HOME>/server/apache/conf/httpd.conf`

The httpd.conf file is used by the web server at start time. The httpd.conf file is made up of individual directives. Directives are instructions which the web server follows for general operation. Any and all uncommented directives found in the httpd.conf file are added to the web servers running configuration on startup. The httpd.conf file also makes reference to two other configuration files needed for Build Forge operation. The first of the two is the php.ini file. This configuration file instructs the Web Server on what php capabilities should be available at runtime. The php.ini file also contains key directives needed for Build Forge management.

- File Location for php.ini reference
  - Windows: `<BF_HOME>\Apache\php\php.ini`
  - Unix/Linux: `<BF_HOME>/server/apache/conf/php.ini`

The second of the two httpd.conf file references is the ssl.conf file. This file instructs the Web Server about the SSL configuration to include at runtime. This file contains key directives necessary for listening for https requests and managing secure connections.

- File Location for ssl.conf reference
  - Windows: `<BF_HOME>\Apache\conf\ssl\ssl.conf`
  - Unix/Linux: `<BF_HOME>/server/apache/conf/ssl/ssl.conf`

2.) What Directives are required for Build Forge Operation?

Key Directives in the httpd.conf file:

- ServerRoot – Identifies the Web Server Installation root directory. Example: `ServerRoot "/opt/ibm/buildforge/server/apache"`
- DocumentRoot – responsible for identifying the root directory containing web content (html / PHP / JavaScript …)
  Example:
  `DocumentRoot "/opt/ibm/buildforge/webroot/public"`
- PHPIniDir – Identifies the location of the php.ini file. The php.ini file contains PHP server configuration information needed by the Web server on startup. Instructs the Web Server where to find the PHP server configuration for serving PHP web content.
  Example:
  `PHPIniDir "/opt/ibm/buildforge/server/apache/conf"`
- LoadModule php5_module – Instructs the server to load the php5 module required for Serving PHP content. The directive identifies the location of the php5 library on the file system (.dll or .so)
  Example: LoadModule php5_module modules/libphp5.so (or .dll)

- Listen – Instructs the server to listen on a given port for HTTP or HTTPS traffic.
  Example: Listen 0.0.0.0:80
- ServerName – Used for redirection URLs
  Example: ServerName localhost:80
- AddType – Instructs the server to listen and serve web requests for identified additional file extensions (in addition to html, htm, ...)
  Example: AddType application/x-httpd-php .php
- LoadModule ssl_module – Instructs the Server to load the SSL module utilized by https requests to the web server.
  Example: LoadModule ssl_module modules/mod_ssl.so (or .dll)
- <IfModule ssl_module> …. </IfModule> - Instructs the server to include any enclosed directives if the SSL Modules is loaded.
  Example:
  ```
  <IfModule ssl_module>
    Include conf/ssl/ssl.conf
    SSLRandomSeed startup builtin
    SSLRandomSeed connect builtin
  </IfModule>
  ```

Key Directives for the php.ini configuration file

- Memory_limit – Instructs PHP to use a maximum of the defined value (Can result in errors and is often increased to 512 or greater)
  Example: `memory_limit = 512M`
- session.save_path – Instructs PHP where to save session information for user http sessions
  - The Build Forge UI uses a three part scheme for controlling user sessions
    - Client Browser cookie
    - PHP session
    - DB entry

  Example: `session.save_path = “C:\IBM\buildforge\tmp”`
- include_path – Instructs PHP where to look for included modules
  Example:
  `include_path=".;C:\IBM\buildforge\Apache\php\pear”`
  extension_dir – Instructs PHP where to find extensions crucial for php extended usage and integration. This directory will contain the database specific extensions needed for build forge usage
  Example:
  `extension_dir = “C:\IBM\buildforge\Apache\php\ext”`
- extension – Instructs PHP which extensions to load on startup
  Examples:
  ```
  ;extension=php_ibm_db2.dll
  ;extension=php_mssql.dll
  ;extension=php_mysql.dll
  ;extension=php_mysqli.dll
  ;extension=php_oci8.dll
  ```
  Example for PHP Open SSL Extension:
  ```
  extension=php_openssl.dll
  ```
- file_uploads – Instructs PHP to allow or block file uploads to the server
  Example: `file_uploads = On`
- upload_tmp_dir – Instucts PHP where to save uploaded files locally on the server
- Example: `upload_tmp_dir = "C:\IBM\buildforge\tmp"`

- upload_max_filesize – Instructs PHP to limit the maximum file size for uploads to the server
  Example: `upload_max_filesize = 2M`
- default_charset – Instructs PHP to default any character encoding to the defined value
  Example: `default_charset = "utf-8"`
- error_log – Instructs PHP where to publish any errors encountered during startup and web interaction
  Example: `error_log = "C:\IBM\buildforge\Apache\logs\php_error.log"`

Key Directives for the ssl.conf configuration file

- SSLEngine – Instructs the SSL engine to be enabled or disabled for any or all virtual hosts controlled by the Web Server
  Example: `SSLEngine on`
- Listen – Instructs the Web Server to listen for https traffic on the specified port
  Example: `Listen 0.0.0.0:443`
- SSLCertificateFile – Instructs the Web Server where to find the SSL Certificate file
  Example:
  `SSLCertificateFile ../keystore/buildForgeCert.pem`
- SSLCertificateKeyFile – Instructs the Web Server where to find the server Private Key file
  Example: `SSLCertificateKeyFile ../keystore/buildForgeKeyForApache.pem`
- SSLCACertificateFile – Instructs the Web Server where to find the Certificate Authority files
  Example: `SSLCACertificateFile ../keystore/buildForgeCA.pem`
- <VirtualHost *:443> …. </VirtualHost> - Instructs the Web Server to apply SSL configuration to all virtual hosts listing on port 443
  Example Build Forge Directives within <VirtualHost *:443>
  - DocumentRoot "C:/IBM/buildforge/webroot/public"
  - ServerName localhost:80
  - ServerAdmin support@buildforge.com
  - ErrorLog logs/error_log
  - TransferLog logs/access_log
  - ………

3.) What Log Files Contain Specific Build Forge Information

There are multiple Log files within the Web Server that contain Build Forge Specific information. The Build Forge Management Console Web Site generates Apache http request/response, SSL Connection and PHP request/response information. This means multiple logs files will be touched when accessing Build Forge Site Specific locations.

Below are the log files that should be monitored according to file size

- Unix/Linux: `<BF_HOME>/server/apache/logs/*`
- Windows: `<BF_HOME>/Apache/logs/*`
    - access.log
    - access_log
    - error.log
    - error_log
    - php_error.log
    - ssl_request_log

4.) Manually Starting and Stopping the Web Server Component

The following information describes how Apache or IHS Web Server platforms can be manually started on Build Forge Systems. There are situations where debugging can be easier when starting individual Components on the Build Forge server. Before starting individual Build Forge components manually, ensure all running Build Forge Components are stopped.

- Startup and Stop options for IHS and Apache Web Servers

- Windows Apache
    - `<BF_HOME>\Apache\bin\ httpd.exe`
    - To Stop, terminate any and all child/parent httpd.exe processes
- Windows IHS
    - `<IHS_Install_Root>\bin\httpd.exe`
    - To Stop, kill the windows running the process
- Unix/Linux Apache (Start and Stop Aache)
    - `<BF_HOME>/server/apache/bin/apachectl start`
    - `<BF_HOME>/server/apache/bin/apachectl stop`
- Unix/Linux IHS
    - `<IHS_Install_Root>/bin/apachectl start`
    - `<IHS_Install_Root>/bin/apachectl stop`

Application Server #2 (Services Layer)

1.) Install / Configuration / Web Application / Logs

- **WebSphere (WAS)**
  - Install
    - The Build Forge Content installed under WebSphere will exist within the deployed scope. The Following example demonstrates where the rbf-services, BuildForgeHelp and rafw "ear and war" files are typically deployed.
    - Install Locations:
      "profiles/standAlone/installedApps/standalone/rbf-services.war.ear"
      "profiles/standAlone/installedApps/standalone/BuildForgeHelp_war.ear"
      "profiles/standAlone/installedApps/standalone/rafw_war.ear"
  - Configuration (This section assumes being logged into WebSphere Application server Admin console)
    - WARNING!!! Please reference the Latest Installation Guide or Online help for any configuration updates or changes to Build Forge Deployment on WebSphere Application Server
    - When working with WebSphere Configurations, please ensure all configured objects are created at the Build Forge Application Server Scope. This ensures all configured objects are available to the Build Forge Server at runtime.
    - Configure Application server Ports
      - Application Servers -> server1 -> Ports
        - WC_defaulthost
          - 9080
        - WC_defaulthost_secure
          - 9443
    - Configurable options within the rbf-services Application
      - Enterprise Applications -> rbf-services_war
        - Context Root For Web Modules
          - Web Module
            - A Services Layer Login Servlet
          - URI
            - rbf-services.war,WEB-INF/web.xml
          - Context Root
            - /rbf-services
        - Initialize Parameters for Servlets
          - Service Layer Login Servlet
            - port 3966
            - sslPort 49150
        - Shared Library References (CONFIRM SCOPE)
          - Service Layer JDBC Library reference
          - RBF_JDBC_LIBRARY
          - Path to Vender JDBC Driver
      - WebSphere Variables -> RBF_JDBC_DRIVER_PATH
        - Path to Vender JDBC Jars (DB2 Example)
          - /home/build/sqllib/java
      - Shared Libraries -> RBF_JDBC_LIBRARY
        - Ensure Scope is accurate

- cells:standAlone:nodes:standAlone:servers:svr1
- Class Path (db2 examples)
  - ${RBF_JDBC_DRIVER_PATH}/db2jcc.jar
  - ${RBF_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar
- Web Applications
  - Within the Installed *.war.ear directories resides the Build Forge Deployed Web Applications. There are a few key files within the deployed Content that are worth noting
    - FlexHelper binary (Platform and OS Specific)
      - ./rbf-services_war.ear/rbf-services.war/bin/*
      - This is the Flexlm License utility used by Build Forge to pull Flexlm licenses from a Rational License Server
      - This binary must be started for the Services Layer to pull a license.
    - buildforge.conf (Contains BF configuration info)
      - ./rbf-services_war.ear/rbf-services.war/WEB-INF/classes/buildforge.conf
      - This file contains database connection information used on startup
      - The file also contains descriptive information about the services port and other URL's identifying location of the Services Layer
- Logs
  - WebSphere Application Server logs will be generated in four significant files for Build Forge usage
    - (SystemOut, SystemErr and start/stopServer)
  - Typical Log File location: (Example standAlone install)
    - /WAS/70/AppServer/profiles/standAlone/logs/server1
  - SystemOut.log and startServer log are worth investigating when experiencing any unusual license or startup behavior.

- **Tomcat**
  - Install
    - Build Forge by default will Install Tomcat to:
      - Windows:
        - `<BF_HOME>\apache\tomcat`
      - Linux:
        - `<BF_HOME>/server/tomcat`
  - Configuration
    - WARNING!! The Tomcat Configuration information below is for reference only. This Build Forge Installation will perform any necessary configuration needed in Tomcat. This information is for management and troubleshooting purposes if needed.
    - Tomcat will default many Service Layer Configuration Options after installation.
    - Web Context Root defaulted on webapp deployment
      - /rbf-services
      - <tomcat_home>/webapps/rbf-services
    - Application Server Ports

            - <tomcat_home>/conf/server.xml
                - Non SSL
                    - <Connector port=8080 .../>
                - SSL
                    - <Connector port=8443 .../>
        - Application Server Host Name
            - <tomcat_home>/conf/server.xml
                - <Engine ... defaultHost="localhost" .../>
                - <Host name="localhost" ..../>
        - To Configure Web Application Specific settings
            - <tomcat_home>/webapps/<webapp_dir>
                - Example ./webapps/rbf-services/*
        - Build Forge Service Layer Configuration Options (web.xml)
            - ./rbf-services/WEB-INF/web.xml
                - Set Service Layer Ports (SSL and Non-SSL)
                    - <param-name>port</param-name>
                    - <param-value>3966</param-value>
                    - <param-name>sslPort</param-name>
                    - <param-value>49150</param-value>
        - ./rbf-services/WEB-INF/classes/buildforge.conf
            - Set any Build Forge parameters accurately for the Services layer to contact necessary runtime resources (db user/pass server hostname and ports)
    - Web Applications
        - All Web Applications will reside under
            - Windows:
                - `<BF_HOME>\apache\tomcat\webapps`
            - Linux:
                - `<BF_HOME>\server\tomcat\webapps`
        - Build Forge and RAF specific Web Applications
            - Services Layer
                - rbf-services (rbf-services.war)
            - rafw Environment Generation Wizard
                - rafw (rafw.war)
            - rafw Remote UI Services
                - rafservices (rafservices.war)
            - Build Forge Help
                - BuildForgeHelp (BuildForgeHelp.war)

    - Logs
        - `<tomcat_home>/logs/catalina.<date>.log`
        - `<tomcat_home>/logs/rafw.<date>.log`
        - Each time build forge is restarted, a new Log file is generated.
        - Please remove or cleanup any unwanted log files.
        - The logs directory is worth scheduling for cleanup for files older than a desired date

2.) Manually Starting and Stopping the Application Server Component

- WebSphere (This will stop and start the Dedicated Application Server)
  - **WARNING: This procedure will not stop the individual Applications!!!**
  - Windows (start / stop)
    - `<WAS_HOME>\profiles\<profile>\bin\startServer.bat <svrName>`
    - `<WAS_HOME>\profiles\<profile>\bin\stopServer.bat <svrName>`
  - Unix/Linux (start / stop)
    - `<WAS_HOME>/profiles/<profile>/bin/startServer.sh <svrName>`
    - `<WAS_HOME>/profiles/<profile>/bin/stopServer.sh <svrName>`
  - Example:

    -
      ```
      /WAS/70/AppServer/profiles/standAlone/bin/stopServer.sh server1
      ADMU0116I: Tool information is being logged in file
      /WAS/70/AppServer/profiles/standAlone/logs/server1/stopServer.log
      ADMU0128I: Starting tool with the standAlone profile
      ADMU3100I: Reading configuration for server: server1
      ADMU3201I: Server stop request issued. Waiting for stop status.
      ADMU4000I: Server server1 stop completed.
      ```
- Tomcat (This will stop and start the Tomcat Application Server)
  - **WARNING: This procedure will not stop the individual Applications!!**
  - Windows (start / stop)
    - Before manually stopping and Starting Tomcat, you must copy <tomcat_home>\bin\catalina.bat.template to <tomcat_home>\bin\catalina.bat
    - Make the following changes in catalina.bat
      - `set CATALINA_HOME=<BF_HOME>\Apache\tomcat`
      - `set JAVA_HOME=<BF_HOME>\ibmjdk`
    - After modifying the file, ensure all other Build Forge Components are stopped
    - Start Server with the following command:
      - <tomcat_home>\bin\startup.bat
      - A new shell will open which must not be closed
      - You can further startup Apache Web Server and the Build Forge Engine
    - You can stop the server with:
      - <tomcat_home>\bin\shutdown.bat
    - Example Startup:
      ```
      C:\IBM\bf_mssql\Apache\tomcat\bin>startup.bat
      Using CATALINA_BASE:   C:\IBM\bf_mssql\Apache\tomcat
      Using CATALINA_HOME:   C:\IBM\bf_mssql\Apache\tomcat
      Using CATALINA_TMPDIR: C:\IBM\bf_mssql\Apache\tomcat\temp
      Using JAVA_HOME:       C:\IBM\bf_mssql\ibmjdk
      C:\IBM\bf_mssql\Apache\tomcat\bin>
      ```
  - Linux/Unix
    - Before manually stopping and starting Tomcat Application server, you must copy <tomcat_home>/bin/catalina.sh.template to <tomcat_home>/bin/catalina.sh
    - Add the following lines to the catalina.sh script
      - JAVA_HOME=<BF_HOME>/ibmjdk
      - CATALINA_HOME=<BF_HOME>/tomcat
    - After modifying catalina.sh ensure all build forge components are stopped
    - Start the Tomcat Server with the following command:
      - <tomcat_home>/bin/startup.sh

- Look for new processes running via catalina and or java binaries running from the JAVA_HOME location
- You can now startup the Apache Web Server and Build Forge Engine Components

▪ You can then stop the server with:
  - <tomcat_home>/bin/shutdown.sh

▪ Example startup:

```
[root@bfSvr bin]# ./startup.sh
Using CATALINA_BASE:   /opt/buildforge/server/tomcat
Using CATALINA_HOME:   /opt/buildforge/server/tomcat
Using CATALINA_TMPDIR:/opt/buildforge/server/tomcat/temp
Using JAVA_HOME: /opt/buildforge/server/ibmjdk
Using JRE_HOME:       /opt/buildforge/server/ibmjdk
```

Management Console #3

1.) Manifest / Server Test / Max Processes / Max Jobs

Manifests:
Manifests represent the container for the story you tell about the server using the collector. The selector reads from this container, and collectors write to it. There is no communication between the collector and selector directly. As a result the selector is dependent upon the information in the manifest being accurate and up to date. There are a number of global Admin > System values which control how up to date the manifest is.

Testing a server
To perform a diagnostic test:

- Click Servers–> ServerName.
- Click Test Connection.
- A server connection test is performed and the server manifest is updated.

Configuring system settings for the management console
Evaluate these system settings before you start creating server definitions and running jobs to decide whether the settings can benefit your environment. Your usage pattern determines the values of these settings. This section helps you determine those values.

SMTP Server: This setting determines which mail server Rational Build Forge uses to send outgoing mail notifications. If your management console is in a network behind a firewall, the value of this setting should be the mail relay server for the network, and you should discuss the setting with your network administrators.

Run Queue Size: This setting controls how many jobs the management console attempts to run at any given time. The default value is 3, which is a fairly conservative number. If your console has four processors, a value of 3 might be too small. Consider that none of the Rational Build Forge processes is likely to use 100% of any processor. If you do not run many threaded steps in your jobs, the highest processor utilization you will see will be about 25% for any given job. With such low utilization of four processors, you could increase your Run Queue Size. Plan on one processor for the Rational Build Forge system processes, and then divide the remaining number of processors by the expected processor utilization. Next, reduce the size slightly to ensure you do not overload the management console. With the above example, set the Run Queue Size to 10. (Four processors minus one for management console system processes is 3. Divide 3 by 25% to get 12. Add a little safety room, and 10 is an adequate setting.)

WARNING: Setting this value too low limits the ability of your console to run projects in parallel. Setting this value too high might have a negative impact on your system performance. Using an incremental approach has proven successful in the past. You could start at 5 and step up by 5 to determine the best value for your use.

Max Console Procs: This setting controls the maximum number of processes that the management console runs at one time. If you set this value too low, your Run Queue Size setting becomes ineffective. If you set this value too high, the stability of the computer that runs your management console might be affected. A good initial value for this setting is 10 + Run Queue Size + Max Simultaneous Purges. So for our earlier examples, set this value to 30.

As of version 7.1.2, only buildforge.exe and bfproject.exe are considered processes started by the console. For optimal behavior, use a value at least 1 higher than Run Queue Size, which is covered next. By default, this setting is 25.

Max Simultaneous Server Tests: As of Rational Build Forge version 7.1.2, server tests are completed through the Java services layer. As such, the tests have a much smaller footprint on the system resources. Consequently, you can use a number for this setting larger than the default value of 6. Typically, you need a higher number of concurrent server tests if your environment has a large number of computers restart often. In this case, you require constant server tests to tell Rational Build Forge when the servers are back online.

2.) Managing System Messages

AutoClean settings: The AutoClean settings determine how long to retain system messages of different types. These are the settings and the related messages:
AutoClean Error Log Days: Error messages help determine faults within the management console. These messages typically account for less than 1 percent of the total message count. The default value is 0, indicating the system never deletes these messages. Because the impact of this message type is so low, leave the value set at 0.
AutoClean Warning Log Days: Warning messages can help determine faults or decide when to perform maintenance. The warning messages account for around 10 percent of the overall message count. By default, these messages are also kept for 120 days; however, the impact is much less than the information messages.
AutoClean Audit Log Days: Audit messages can let you know when security settings have been changed. Keep them longer. In a typical system, audit messages account for around 20 percent of the total message count. The default setting is 365 days.
AutoClean Info Log Days: Info messages act as a log of activity on the system. These messages will be quite numerous. Typically, you do not refer to these messages after a few days. Information messages account for approximately 70 percent of the overall message count. This setting has the most impact in your environment. The default count is 120 days, which is too long for most installations. Typically, you should keep 7 to 14 days of information.

All of these messages help you understand what is going on in your management console, but the default settings are for a large organization with a high-performance database server with requirements to keep messages for long periods. Typically, you do not require that any of these messages be saved for more than 30 days. Perhaps you only require them for a few days.

WARNING: The AutoClean log settings are set very high by default. As a result, Rational Build Forge tends to store a large amount of the system messages within the database. Modify these numbers to fit your company's use case.

WARNING: Changing these settings can significantly impact system performance. The AutoClean logs will attempt to delete any and all messages older than the set number of days. This means a large scale purge could result in poor system performance and reliability issues.

It is strongly recommended to incrementally reduce the time to keep messages. (Example: Start with a high number of days allowing the purge to be throttled over time.)

It is also recommended that you make each AutoClean number of days different. This assists in distributing purge scheduling for different types of message information.

Database Size Threshold and Database Size Threshold Notification: These settings serve as an early warning system. Rational Build Forge sends a notice to the address specified in Database Size Threshold Notification when the database exceeds the disk size in the Database Size Threshold setting.

The Database Size Threshold setting has a default value of 2 GB, which is too low for most Rational Build Forge environments. Set the value to 80% of the maximum amount of the disk space available to the data files of the database. Using this value provides time to take corrective actions before running out of disk space.

3.) Managing Purging at Project Level

Max Simultaneous Purges: This setting controls how many purges can run at the same time. Adjust this value to ensure that purges are completed in an acceptable period of time without affecting the performance and usability of the management console. The more purges that run at the same time, the busier and less responsive your management console is. If you have a large computer hosting your management console, this setting will not matter as much. Similarly, if you schedule your purges to run during off hours, the performance is not as large an issue. However, if your management console has international users and is used at all hours, set this value to a smaller number to maintain usability of the management console. Be careful not to set the value so small that your purges cannot be completed faster than new jobs are added. For example, if you are running 20 jobs an hour and your jobs take 12 minutes to purge each, you would need to run at least four purges concurrently. Adding a safety factor to this value, for this example, start at 10 Max Simultaneous Purges.

WARNING: Setting this number higher allows a larger number of purges to occur concurrently; however, it also reduces build performance.

Configuring classes to purge schedules and start related projects
Rational Build Forge uses classes to determine different attributes of both running and completed jobs. For example, you can start a project as a chain when changing into or out of any given class. You can use this feature to have Rational Build Forge deploy built software to be tested. Classes also determine how soon builds are purged.

Consider defining a few basic types of classes: short-term, mid-term, and long-term.

You might assign a short-term class to a build that happens every day to send a status email or clean files on build servers. You might assign a short-term class to purge jobs after one day. Frequent purging is useful for jobs that are not required for logging purposes and for jobs that do not generate important output.

A mid-term class might have a purge time of a few weeks. Use this class for daily builds that you keep an eye on and perhaps check for changes after a few days. However, do not use this class for builds that you need to keep forever.

A long-term class might be set to never purge. For example, if you released software to the public that was built from your management console, you would likely want to keep that build forever. Setting the build to such a long-term class is one way to accomplish that.

Configure classes shortly after you start running builds to maintain order in your completed jobs and to keep completed jobs from cluttering your database.

4.) Managing Step log information to reduce Database size

There are two commonly used methods for managing step log information, purging, and utilizing bfbomexport.

The best way to purge old log entries is to set up a class to do so. This class can be set up to meet your criteria - for example purge all builds older than 30 days. If the class does not have Everything or Console Data specified for the items it will purge the build becomes archived. You can then clear out the archived builds through the Archived tab on the Jobs page.

You will want to determine what your purge criteria will be - i.e. how often you would like to remove the builds. In some cases - such as continuous integration builds - you may only want to keep the last failed build. In this case you will use a very aggressive purge policy. In other cases - such as a release - the build needs to be maintained for possibly years. We cannot make recommendations on how often you purge as you know how long you need to keep certain builds around.

The class mechanism is specifically designed to automate the process detailed above. When a purge check occurs it will look at each build for a class for each individual project. If the files to be deleted by the class not set up for Everything, logs, logs and files, or console data you will have build records left in the database under the Archive tab. Archive builds are no longer considered during the automatic purge process and must be manually purged.

There are various logs and messages BF keeps in the underlying database and it is possible to change length of time these are kept in order to systematically reduce the size of your database.

However in large corporations where some data needs to be kept for long periods of time over many years or even forever, you will need to be able to export logs out for archiving.

The bfbomexport tool has had **log**\BOM export tool as well which can be used for this purpose - bfbomexport.

Usage: bfbomexport [-f filename] [-p projectId |-P projectName] [-b buildId |-t buildTag] [-H]

options :
-f : file to write the output to.
-p : project Id of the build. -p or -P must be specified.
-P : project Name of the build.
-b : build Id to export the BOM of. -b or -t must be specified.
-t : build tag to export the BOM of.
-L : include Step logs.
-h : this help message.

Keeping a low revolving database size can be very important to BF performance. Disk I/O can be dramatically affected with the number of concurrent builds. Considering that at any given time BF can be reading or writing to the DB, which is using the DB pipeline, the size of the DB directly affects the BF performance. The larger a DB becomes the more expensive write operations to the DB file become.

5.) Know what is scheduled

The scheduler in 702\7.1.1.x uses a simple algorithm to determine if a schedule can be fired or not. Specifically it follows this pattern:

1) Loads all the currently active schedules into memory
2) Tests if the schedule should fire in this month
3) If that passes, tests if the schedule should fire in this day of the month _or_ week day
4) If that passes, tests if the schedule should fire in this hour
5) If that passes, tests if the schedule should fire in this minute
6) If that passes, fire the build

The schedule _has_ to match all five of those - down to the current minute. Now one of the weaknesses of the old scheduler is it does not test if the schedule should fire based on this current minute, or a minute in the past. To that end this is why we determine the entire cycle needs to be done for _all_ currently active schedules in under one minute. In 7.1.2 we use a proper thread, and scheduler class in the services layer to handle each schedule - it is not done in the same iterative fashion as Perl. As such the scalability is much higher, as well as the reliability.

To see more specifically this schedule cycle requires killing the bfsched process, and starting it in its own shell, however the entire Build Forge process does not need to be restarted. The scheduler can run this way as long as needed. That being said two debug variables are needed. This will give us incredible insight into how long it takes for each scheduler cycle to complete.

```
BFDEBUG_SCHED=1
BFDEBUG_SQL_PREPARE=1
```

Example of what we will see in the capture:

A single scheduler cycle will be delimited by a long ----- line:

```
6/7/2011 4:39:10 PM : Sched: 48124:     -----------------------------------
-------------------------------
48124 PRE:[SELECT * FROM bf_engines WHERE bf_engineid=?]
6/7/2011 4:39:25 PM : Sched: 48124:     -----------------------------------
-------------------------------
```

In between is all the stuff that a scheduler normally gets to. If the time stamps for each ---- line exceed a minute, or even come close to a minute then we know the scheduler is being stressed too hard for 7.0.2 and the current workaround the customer is employing to fire off builds if they don't fire can still be used.

```
Sched: 48124: ------------------------------------------------------------------
6/7/2011 4:43:47 PM : Sched: 48124: Processing
48124 PRE:[SELECT * FROM bf_users WHERE bf_userid=?]
48124 PRE:[SELECT * FROM bf_accesscache WHERE bf_userid=0 OR
(bf_userid=? AND bf_groupid=0)]
48124 PRE:[SELECT bf_tzone FROM bf_users WHERE bf_userid=?]
48124 PRE:[SELECT * FROM bf_tzones WHERE bf_zone=?]
6/7/2011 4:43:47 PM : Sched: 48124: a: Month * match on 5
6/7/2011 4:43:47 PM : Sched: 48124: a: Day * match on 7
6/7/2011 4:43:47 PM : Sched: 48124: a: Weekday * match on 2
6/7/2011 4:43:47 PM : Sched: 48124: a: Hour * match on 15
48124 PRE:[SELECT * FROM bf_engines WHERE bf_engineid=?]
6/7/2011 4:44:02 PM : Sched: 48124:     -----------------------------------
-------------------------------
6/7/2011 4:44:02 PM : Sched: 48124: Processing
48124 PRE:[SELECT * FROM bf_users WHERE bf_userid=?]
48124 PRE:[SELECT * FROM bf_accesscache WHERE bf_userid=0 OR
(bf_userid=? AND bf_groupid=0)]
48124 PRE:[SELECT bf_tzone FROM bf_users WHERE bf_userid=?]
```

```
48124 PRE:[SELECT * FROM bf_tzones WHERE bf_zone=?]
6/7/2011 4:44:02 PM : Sched: 48124: a: Month * match on 5
6/7/2011 4:44:02 PM : Sched: 48124: a: Day * match on 7
6/7/2011 4:44:02 PM : Sched: 48124: a: Weekday * match on 2
6/7/2011 4:44:03 PM : Sched: 48124: a: Hour * match on 15
6/7/2011 4:44:03 PM : Sched: 48124: a: Minute */2 match on 44
48124 PRE:[SELECT * FROM bf_cron b WHERE bf_cid=?]
48124 PRE:[SELECT * FROM bf_store where bf_uid=? ORDER BY bf_part]
48124 PRE:[SELECT * FROM bf_store where bf_uid=? ORDER BY bf_part]
48124 PRE:[SELECT * FROM bf_users WHERE bf_userid=?]
48124 PRE:[SELECT * FROM bf_accesscache WHERE bf_userid=0 OR
(bf_userid=? AND bf_groupid=0)]
48124 PRE:[UPDATE bf_cron SET bf_nextrun=0, bf_fired=? WHERE
bf_cid=?]
48124 PRE:[SELECT bf_pid,bf_class FROM bf_builds WHERE
bf_process='B' AND bf_state='C' AND bf_stage != 'Break' ORDER BY
bf_pid]
48124 BF_STATE
/PerlApp/BuildForge/Scheduler.pm(BuildForge::Scheduler):440 [SELECT
bf_pid,bf_class FROM bf_builds WHERE bf_process='B' AND bf_state='C'
AND bf_stage != 'Break' ORDER BY bf_pid]
48124 PRE:[SELECT * FROM bf_engines WHERE bf_engineid=?]
6/7/2011 4:44:18 PM : Sched: 48124: --------------------------------------
---------------------------
```

This part shows the evaluation of one schedule only. You can see where it matches on everything except for minute in the first cycle, and then matches on the minute in the following cycle prompting a build.

In all the debug flags will give us an excellent idea on how efficient the current schedule count really is, and how much stress each schedule really provides on the overall schedule cycle.

6.) Where is your License Server / How is it Configured

Build Forge stores the license server it points to within the database.

Update the License Server from Administration> System> License Server. You can do this because the Web interface still functions even without a license as long as you login as root. However, without a license you cannot run any builds or other operations.

There are two types of licenses in Build Forge: Floating and Authorized.

You can only have one type of license available in your environment since, during startup, the engine picks up a single increment of user licenses, which is either Floating or Authorized.

A Floating license seat is purged automatically during user log off, but an Authorized is not. You can manually purge an Authorized license seat by using **Purge Seat** within the Administration > Users menu.

Note: Using **Purge Seat** on a Floating license seat has the same effect as logging off the user since either action will simply release the license seat.

A brief summary of each license type

- **Floating**: These license seats are best thought of as a first come, first served pool of seats that anyone can use. This allows an oversubscribed model of users to access the server on the assumption that not everyone will be working at the same time.**Authorized**: These license seats represent a fixed number of specific users that can access the system. This model explicitly does not allow the oversubscription model and, as a result, cost less on a per-seat basis. This is similar to an *assigned sea**t*** **logon sc**heme.

Database / Engine Management #4

1.) **Where are the Database Logs (How to increase Debugging in Logs)**
The system stores database debugging information in a db.log file in the Management Console installation directory. You s*hould check t*he size of this file on a monthly basis, and delete it if you need to free space on the console computer. The db.log file is located in the BF home directory.
***** Delete the database log file regularly *****

2.) **Where are the Engine Logs (How to increase Debugging in Logs)**
Log file: BF_INSTALL_DIR/log (or run in foreground)

3.) **What tables should be monitored?**

Tables grow with as the use of BF increases. There are methods to help keep these tables manageable. Below are a few tables to keep an eye on.

Messages Tables

bf_messages - System wide event log messages. The bf_messages table gets translated to allow for full message searches.

bf_messagearg - The table which holds the string arguments to the keys defined in the bf_messages table.

bf_message_translated - This table is used to manage the potentially large, fully-translated system message strings. Messages are pre-translated for query performance and scalability. Each message inthe bf_messages table becomes 10 rows in the bf_message_translated table. There is one row for each language BF currently supports.

NOTE: There are two basic methods to keeping these tables small.

1- Manually truncate the bf_messages table. This is very risky, and not always recommended. There are a number of tables involved which would need truncated.

2- Use the system to purge messages.

bf_store - This table is used to manage large strings that would otherwise overflow the database managers allowed sizes (and to avoid using the CLOB and BLOB SQL data types).

Build Environment Tables

bf_buildenv - Build Environments are per- build or per- schedule workspaces for manipulating environment group data without damaging the original. It is implemented by creating a deep copy of the environment group when the build record is created. This table will grow in proportion to the number of jobs executed and the number of environment variables associated with the project run.

bf_buildenventry - Contains name/value pairs of environment variables associated with a particular build or schedule. These environment variables have been copied from bf_enventry in order to have a working copy where the environment variables can be stored and manipulated independently of the original. This table grows as the number of runs grows, increasing by the number of environment variable entries receive a value.

bf_buildenvopt - Build environment entry options. This table contains the values that will be accepted for a build environment entry of the PULLDOWN type. They were copied from bf_envopt to have them stored so that the user is presented with the same choices from the pull down list on build restart, even if the original environment entry has changed in the meantime. This table will remain small.

bf_results - The bf_results table contains all of the step scope pass\fail information normally found in a build log at the higher level If the bf_results table has rows in which the bf_bid does not exist in the bf_builds table then it has been orphaned. Purging through the user interface will no longer be possible and the rows will need to be manually removed. This will require an orphan removal tool, that can be supplied by BF support.

bf_logs - This is where the step log information is stored. The unique key in this table is a combination of the bf_uid and bf_lid. The bf_log table is linked back to the bf_results table through the bf_uid. If the bf_uid from the bf_logs table no longer has a step listed in bf_results then the row in this table has been orphaned. There could be any number of entries for each bf_uid in the bf_logs table. If there are excessive amounts of orphans, a tool can be supplied to remove them. This table will grow massively if left unmanaged.

bf_bom_manifests - This table is supplemental to the bf_bom table. There is the distinct possibility there are orphaned rows if the bf_logs contains orphans as well.

NOTE: The best way to manage table sizes is to implement a strong purge schedule, utilizing aggressive purging as needed to keep the DB at manageable size.

4.) What is a Schema and why should we care about it.

A database schema is a collection of meta-data that describes the relations in a database. A schema can be simply described as the "layout" of a database or the blueprint that outlines the way data is organized into tables. Schema are normally described using Structured Query Language as a series of CREATE statements that may be used to replicate the schema in a new database.

You can obtain detailed IBM Rational Build Forge schema information on the database by using the bfschema tool.

```
bfschema -g <path to output.html file>
```

bfschema is located in the BF home directory. Once you run the tool the output file can be read using any http compatible browser.

5.) How to predict and monitor a healthy database growth rate

In this section is a high level approach to creating a database Resource Plan using analytical methods. All that is required is to plug in values for simple arithmetic formulas. It is recommended that you consider the use of a spreadsheet to simplify the process of performing the calculations, leaving you free to concentrate on obtaining the input data, developing the model, and validating the results.

Create the Test Schema:

To accurately gather database growth metrics, a new build forge user and schema should be created. In the case of databases like Oracle, care should be taken to ensure that the new schema has its own dedicated table space. This will make gathering growth data less burdensome.

Create a Simulated Data Load:

When possible, actual build output logs should be incorporated into the test project, so as to yield a more realistic Resource Plan. Additional data that should be acquired includes the approximate number of steps in the build project(s) and size of the log output.
This can be accomplished in a couple ways.

Supplied Build Log Output:

Integrate the supplied output log into the step of the test project. Having the agent echo the contents of the log on the remote side will generate a reasonable approximation of the amount of step log content as would be seen in actual usage. This is the more accurate method for simulation of usage, and thus is preferred.

Supplied Build Log Size Estimates:

If you are unable to obtain a previous log output, but have access to a size estimation of log output, you can achieve the same result by creating a dummy file of approximately the same size as the real logs. Again, echoing this data in the agent will generate a reasonable approximation of the amount of step log content as would be seen in actual usage. This method is not ideal and not recommended when actual log output data is available.

Create the Test Project:
The number of steps in the Test Project depends on the data supplied. If a complete Build Log has been supplied, a single step may be sufficient at providing a low-end Resource Plan. If using the alternate method, additional steps and environment groups may need to be added to the Test Project to more effectively simulate actual usage.

Capture the Initial Size:
The size of the database once Build Forge has been installed and the Test Project created will need to be captured for factoring into later metrics. As later metrics will center on database growth, the initial post-install database size will need to be considered to create a more realistic usage projection.

Calculate the Average Growth per Build Run:
To generate the Growth per Build Run, execute the Test Project **N** times. Then capture the new database size, allowing time for the database to write the data to storage. Subtract the Initial Size, and then divide by **N**. This will give you the average growth per individual build run. Once you have this figure, creating the Resource Plan is a relatively simple procedure. The value of **N** should be a sufficient number to create a statistically valid average. This could be a number as low as 50, or as high as 200. In smaller output logs, **N** should be a number large enough to cause a measurable change in database consumption. Also be aware that overly large values of **N** will take an increased time to produce useful metrics.

Create the Resource Plan:
In addition to the data captured, information on individual project build schedules, release schedules, and data retention policies are required to produce a Resource Plan on Build Forge database consumption.

1. Build Schedule: This is how many times per day a specific project is run per day/week. This number will affect the total number of builds run during a specified duration.

2. Release Schedule: Typically a Released build will be locked in the Console, and its data must be taken into account when factoring the Resource Plan.

3. Data Retention Policy: Tuning the purge policy is an easy way to reclaim database resources in the long term, decrease Console page load times, and make a Resource Plan more attractive. However, this should be balanced against the need to retain data.

Using the supplied method in the next section: *Utilities for Database Resource Planning with Build Forge*, create the Resource Plan predictive model for database storage consumption.

5 Utilities for Resource Planning with Build Forge

The Resource Plan can be constructed using the formula below, combined with the collected data to produce a minimum growth prediction for a given duration.

Build Size: The Average Database Growth per Build.
Duration: The length of the metric in days.
Purge Policy: The minimum number of days a build logs to retain.
Archival Cycle: Process of locking one or more build logs for archival retention.
Lock: Manually excluding a build log from the Purge Policy

Collected Data

- **S** = Approximate Build *S*ize in Megabytes.
- **B** = *B*uilds per day.
- **D** = *D*uration in days.
- **P** = Number of days in *P*urge Policy
- **A** = Number of days per *A*rchival Cycle

Calculated Data

- **K** = Minimum number of build logs to be *K*ept per Purge Policy. [**B*****P**]
- **L** = Number of build logs *L*ocked over desired Duration. [**D**/**A**]
- **R** = Number of build logs *R*etained for desired Duration. [**K**+**L**]
- **T** = *T*otal estimated growth of database for desired Duration. [**R*****S**]

Utilize the collected data as follows:

Calculate the number of build logs kept for reference purposes ***K*** by multiplying the builds per day ***B*** by the number of days defined by the Purge Policy ***P***. These build logs are used by developers and QA for diagnostics and historical tracking.

Calculate the number of archived build logs ***L*** by dividing the length of period desired in days ***D*** by the number of days per Archival Cycle ***A***. These build logs will be permanently retained and counted outside of the build logs temporarily retained by the Purge Policy. (For example: 1 year = 365 days, 30 day archival cycle, the result would be 365/30 or 12 builds locked per year.)

Calculate the total number of build logs to be retained ***R*** by adding the number of archived build logs ***L*** with the number of build logs kept for reference purposes ***K***. Within the desired duration, this is the probable maximum number of build logs retained in the database storage.

Calculate the total estimated database growth for the desired duration ***T*** by multiplying the number of build logs retained over the duration ***R*** by the approximate size of database storage consumed per build log ***S***. This will yield an estimated database growth in megabytes.

Use the following worksheet to generate the database resource usage figures. To project for multiple durations, a spreadsheet will be more useful.

Database Resource Usage Worksheet

Collected Data		
Symbol	**Description**	**Value**
S	Approximate Build *S*ize in Megabytes.	**S =**
B	*B*uilds per day.	**B** =
D	*D*uration in days.	**D** =
P	Number of days in *P*urge Policy	**P** =
A	Number of days per *A*rchival Cycle	**A** =

____________________ X _____________________________ =
Builds per Day ***B*** | Number of Days in Purge Policy ***P*** | Build logs to be kept ***K***

____________________ / _____________________________ =
Duration in days ***D*** | Days per Archive Cycle ***A*** | Build logs locked ***L***

_________________________ + ___________________________ =
Build logs to be kept ***K*** | Build logs locked ***L*** | Build logs Retained ***R***

_________________________ X _______________________ =
Build logs Retained ***R*** | Size in MB ***S*** | Database Growth in MB ***T***

Sample Database Resource Usage Worksheet

The worksheet below uses an example Build Forge installation with one project in an aggressive build environment consisting of 5 builds per day, a 10 day informal release cycle, with a 90 day purge policy that produces approximately 200 MB of Oracle database storage per job run. Remember that most Build Forge installations include multiple projects. This worksheet will need to be completed per project being evaluated.

Collected Data		
Symbol	**Description**	**Value**
S	Approximate Build *Size* in Megabytes.	**S = 200 MB**
B	*Builds* per day.	**B** = 5
D	*Duration* in days.	**D** = 365
P	Number of days in *Purge* Policy	**P** = 90
A	Number of days per *Archival* Cycle	**A** = 10

5	X	90	=	450
Builds per Day ***B***		Number of Days in Purge Policy ***P***		Build logs to be kept ***K***

365	/	10	=	36.5
Duration in days ***D***		Days per Archive Cycle ***A***		Build logs locked ***L***

450	+	36.5	=	486.5
Build logs to be kept ***K***		Build logs locked ***L***		Build logs Retained ***R***

486.5	X	200	=	97,300
Build logs Retained ***R***		Size in MB ***S***		Total Database Growth in MB ***T***

In the above example, with duration of 1 year, and the following the policies as defined, the database would experience an estimated 97,300 megabytes of growth.

The following tables provide examples from constructed data using the techniques detailed in the prior section of the document. This data should not be used verbatim. A proper Resource Plan requires metrics from the proposed installation environment to provide an accurate estimate.

Table 2: Database Growth Examples:

Disk Usage	**Oracle**	**Mysql**	**SQLServer**	**DB2**
Initial Db size	22 MB	20 MB	8 MB	93MB
small Job (10 steps)	6 MB	1.8 MB	980 K	860 K
Med. job (50 steps)	35 MB	6 MB	4 MB	4 MB
large job (100 steps)	80 MB	13 MB	8 MB	7.5 MB

This chart shows the estimated database growth with projects of varying sizes.

Please keep in mind that database size is only one factor to consider when choosing a database.

Note: The Build Forge Installation Guide recommends allocating initial data and log file sizes of 500 MB, with an automatic increase of 500 MB on some platforms.

6 REFERENCES

- Build Forge Online Help

- Special Thanks to
  - Robert Haig (IBM Rational Build Forge)
  - Kristofer A. Duer (IBM Rational Build Forge)