



Use Cases for System i Support in IBM Rational Build Forge

By [Leigh Williamson](#)
Rational Distinguished Engineer
Build Forge Chief Architect
IBM Corporation

Level: Intermediate

May 2008

Contents

Contents.....	2
Overview	3
Build Forge architecture.....	3
i5/OS considerations	4
Use case #1: Simple RPG program.....	6
Use case #2: RPG program with multiple modules.....	6
Use case #3: RPG program with physical file support	7
Use case #4: RPG program with display file support	7
Use case #5: RPG program with command support.....	8
Use case #6: RPG program with logical file support	8
Use case #7: RPG program with menu support	9
Use case #8: RPG program with CL support.....	9
Use case #9: Simple COBOL program	9
Use case #10: Simple Java application.....	10
Use case #11: Composite application.....	10
Trademarks.....	11

Overview

This section provides a very brief, high-level description of the IBM® Rational® Build Forge® product architecture, which will help you understand the content of this article before you read the details of the use cases for the product specific to IBM® i5/OS®.

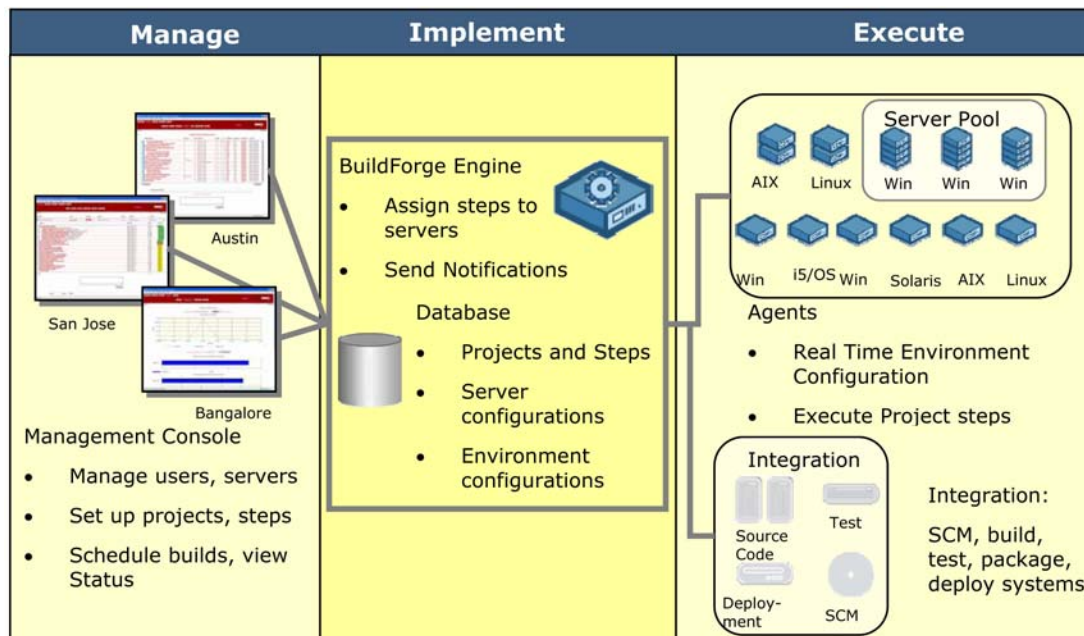
This overview section also includes some background on typical application development for the i5/OS environment. This information is intended to put the i5/OS Build Forge use cases into context with the product and the environment in which the use cases execute.

Build Forge architecture

The Build Forge architecture involves three main components: the user interface, the engine, and the agents (see Figure 1 following). **The user interface** is a Web application that displays in a Web browser. It allows you to define projects and other configurations for controlling the behavior of the process being automated by Build Forge. There is also a plug-in for Eclipse that provides an alternative user interface, as well as a plug-in for Microsoft® Visual Studio. All configuration information that you enter through the user interface is stored in a relational database. You have a choice of several database vendors to use for this repository of Build Forge data.

The **Build Forge engine** reads the configuration data from the database and acts on it. Most typically, this means that it dispatches commands to remote agents that are executing on the systems where the real work occurs. The dispatching algorithm is sophisticated: it supports the selection of a target system from a pool of machines based on the system characteristics required for the task to be executed. The engine also performs other functions, such as send e-mail notifications when configured tasks are completed, and store task result information coming back from the agent performing the activity.

Figure 1. Build Forge architecture



The **Build Forge agent** receives instructions from the Build Forge engine, and then executes those instructions on the system on which it is installed. The agent is very small and simple. It acts as a remote extension of the engine, and does not read or write to the Build Forge database directly. In order for the agent to execute a particular task successfully, it is typical that environment variables specific to that task must be used. Those environment settings are part of the project configuration stored in the database and transferred to the agent as part of the instructions for each step in the process.

Another element of the product is the set of **Build Forge adaptors**. An adaptor is a set of reusable instructions for integrating with other products. There are Build Forge adaptors for many source control products. You can also use an adaptor to supply customized result data back to the engine, store it in the database, and enable subsequent analysis and reporting.

The components of Build Forge are designed and intended to be executed on different systems. The database may be located on one computer, the engine on a separate system, and the agents almost always execute on separate systems.

The initial release of Build Forge support for i5/OS involves the delivery of a Build Forge agent that executes on IBM® iSeries® systems. This initial version of the agent runs under the Portable Application Solutions Environment (PASE) environment on i5/OS. Most tasks necessary for building applications on i5/OS are accessible from the PASE environment. However, the fact that this first release of a Build Forge agent for i5/OS runs under PASE is important to keep in mind when planning and defining the automation of processes targeted for the iSeries platform.

i5/OS considerations

Build practices for building applications on the i5/OS environment involve some unique aspects that are specific to the iSeries platform. The **Integrated Language Environment (ILE)** is a set of tools and associated system support designed to enhance program development on the iSeries system. Runnable objects produced using ILE include ILE programs, and also service programs (which are similar to subroutine libraries, DLLs, or procedure libraries).

The process for creating ILE programs or service programs gives you flexibility and control when you design and maintain applications. The process includes two steps:

1. Compile source code into modules
2. Bind modules into an ILE program or service program. Binding occurs when the Create Program (CRTPGM) or Create Service Program (CRTSRVPGM) command is run.

Languages that you can use to compile ILE modules include the following:

- ILE RPG
- ILE COBOL
- ILE C
- ILE C++

For example, the Create RPG Module (CRTRPGMOD) command compiles RPG source code to create a module object (*MODULE).

There are commands for compiling the various High Level Language (HLL) source code, as shown in Table 1.

Table 1. Compile commands

Command name	Descriptive name
CRTCBLPGM	Create COBOL Program
CRTCLPGM	Create CL Program
CRTCPGM	Create C Program
CRTRPGPGM	Create RPG Program

In addition to HLL source, there are several other elements that can make up an application for iSeries, including:

- Command source
- Display file source
- Menu source
- Help source
- UIM source
- Message source
- And so on

A traditional means of describing data attributes for these elements (such as the names and lengths of records and fields) is to specify the data attributes in the application programs themselves. However, a convenient and powerful alternative is available on iSeries. Data description specifications (DDS) describe data attributes in file descriptions that are external to the application program that processes the data. The file types that use DDS are:

- Physical and logical files
- Display files
- Printer files
- Message files

There are commands for creating these various elements of the application from DDS. Table 2 shows examples.

Table 2. Create commands

Command name	Descriptive name
CRTCMD	Create Command
CRTDSPF	Create Display File
CRTPRTF	Create Printer File

All of the Control Language (CL) commands listed in the previous tables are accessible from the PASE environment by using the system PASE utility. By default, any spooled output produced by the command is written to standard output, and any messages sent by the command are written to standard output or standard error (depending on whether the CL command sent an exception message).

You need to set the ILE environment variable `QIBM_USE_DESCRIPTOR_STDIO` to `Y` or `I` (so that the i5/OS PASE runtime and ILE C runtime use descriptor standard I/O) to avoid

unpredictable results. This is done by default in the i5/OS jobs that the QP2TERM program uses to run i5/OS PASE shells and utilities.

There are several source code control products that support the iSeries system. Leading source control products for iSeries include:

- Aldon
- MKS
- Softlanding

This first set of i5/OS use cases for Build Forge will not begin with a source control step, as is typically the case in use cases for other platforms. All of these use cases assume that the program source has been extracted from its control library and made accessible before the use case begins. While adaptors for i5/OS source control products are under development, they are not included in the release of support for i5/OS in Build Forge Version 7.0.1 or earlier.

Each of the following use cases represents a single, separate Build Forge Project. The steps described in the use case represent individual Build Forge Steps that make up the Build Forge Project.

Use case #1: Simple RPG program

This use case starts with a very basic RPG program. The program source is stored in a source physical file, and there is only one module involved. The resulting program is stored in a library that must be created as part of the build steps.

The individual steps that are sent to the i5/OS Build Forge agent running under PASE are:

1. Execute the command to compile the RPG source and bind it into an ILE program object (*PGM). For example:

```
system -boe "CRTRPGPGM PGM(FLGHT400/BCUST)
             SRCFILE(FLGHT400/QRPGSRC) SRCMBR(BCUST) "
```

This command calls the compiler for ILE RPG to create a program named BCUST. The source program is in member BCUST of source file QRPF SRC in library FLGHT400. A compiler listing is created.

Use case #2: RPG program with multiple modules

This use case is very similar to the previous one, except that the source code involves several modules. Each source module is compiled into an ILE module object, and all of the compiled modules are bound into an ILE program.

The steps for this use case are:

1. Execute the command to compile the first RPG source into an individual ILE module object. For example:

```
system -boe "CRTRPGMOD MODULE(FLGHT400M/NFS402)
             SRCFILE(FLGHT400M/QRPGLESRC) "
```

This command calls the compiler for ILE RPG to create a module named NFS402. The source module is in source file QRPGLSRC in library flght400m. The resulting *MODULE object will be created in library FLGHT400M. A compiler listing is created.

2. Execute the command to compile the subsequent RPG source into individual ILE module objects. For example:

```
system -boe "CRTRPGMOD MODULE(FLGHT400M/NFS404)
            SRCFILE(flght400m/QRPGLSRC) "
```

```
system -boe "CRTRPGMOD MODULE(FLGHT400M/NFS405)
            SRCFILE(flght400m/QRPGLSRC) "
```

3. Execute the command to bind the module objects into a runnable program. For instance:

```
system -boe "CRTSRVPGM SRVPGM(flght400m/NFS400)
            MODULE(flght400m/NFS402 flght400m/NFS404
            flght400m/NFS405)
            TEXT('Flight 400 Information Procedures')"
```

Use case #3: RPG program with physical file support

This use case adds a step for executing the command to process DDS for physical file support by the application.

The steps for this use case are:

1. Execute the command to create a physical file (this step should be first if your RPG source uses the physical file, or an error will occur). For example:

```
system -boe "CRTPF FILE(FLGHT400/AIRLINET)
SRCFILE(FLGHT400/QDDSSRCF) SRCMBR(AIRLINE) "
```

This command creates a physical file and physical file member, both named AIRLINET in the FLGHT400 library. The file and its member are created from the AIRLINE source member of the QDDSSRCF source file in the same library.

Note: If you need to create a Logical file (CRTLF), follow the step to create a physical file.

2. Execute the command to compile the RPG source into individual ILE module objects.
3. Execute the command to bind the module objects into a runnable program.

Use case #4: RPG program with display file support

This use case is similar to use case #3, except that the DDS describes a display file rather than a physical file.

The steps for this use case are:

1. Execute the command to create a display file from DDS source. For example:

```
system -boE "CRTDSPF FILE(FRS000DF)
            SRCFILE(FLGHT400/QDDSSRCD) "
```

This command creates a display device file named FRS000DF, which is stored in the current library using the source file named QDDSSRCD that is stored in the FLGHT400 library.

2. Execute the command to compile the RPG source into individual ILE module objects.
3. Execute the command to bind the module objects into a runnable program.

Use case #5: RPG program with command support

The program for this use case includes the requirement for command support.

The steps for this use case are:

1. Execute the commands necessary to create other modules for the program, such as display files or physical files.
2. Execute the command to create a command from the application source. For example:

```
system -boE "CRTCMD CMD(FLGHT400/USECLEAR)
            PGM(FLGHT499/USECLEAR)
            SRCFILE(FLGHT400/QCMDSRC) "
```

The command named USECLEAR is created from the source file QCMDSRC. It is a valid command when entered in a batch input stream, when compiled in a control language program, when entered interactively, or when passed to the QCMDEXC program.

3. Execute the command to compile the RPG source into individual ILE module objects.
4. Execute the command to bind the module objects into a runnable program.

Use case #6: RPG program with logical file support

This use case adds a step for executing the command to process DDS for logical file support by the application.

The steps for this use case are:

1. Execute the command to create a logical file. For example:

```
system -biOE "CRTLF FILE(FLGHT400/AGENTST)
            SRCFILE(FLGHT400/QDDSSRCF) SRCMBR(AGENTSL) "
```

Note: If you need to create a Logical file (CRTLF) it would need to follow the step to create a physical file

2. Execute the command to compile the RPG source into individual ILE module objects.
3. Execute the command to bind the module objects into a runnable program.

Use case #7: RPG program with menu support

This use case adds a step for executing the command to process DDS for menu support by the application.

The steps for this use case are:

1. Execute the command to create a menu. For example:


```
system -boe "CRTMNU MENU(FLGHT400/FRSMANT)
              TYPE(*DSPF) DSPF(FLGHT400/*MENU)
              MSGF(FLGHT400/*MENU) "
```
2. Execute the command to compile the RPG source into individual ILE module objects.
3. Execute the command to bind the module objects into a runnable program.

Use case #8: RPG program with CL support

This use case adds a step for executing the command to process DDS for CL support by the application.

The steps for this use case are:

1. Execute the command to create a menu. For example:


```
system -boe "CRTCLPGM PGM(QGPL/VERIFYCL)
              SRCFILE(QGPL/QCLSRC) TEXT('Test CL Program') "
```
2. Execute the command to compile the RPG source into individual ILE module objects.
3. Execute the command to bind the module objects into a runnable program.

Use case #9: Simple COBOL program

This use case involves a program whose source language is COBOL rather than RPG.

The steps for this use case are:

1. Execute the COBOL compiler, pointing it to the source code module. For example:


```
system -boe "CRTCLPGM PGM(QGPL/VERIFY)
              SRCFILE(QGPL/QCBLLESRC) SRCMBR(VERIFY)
              TEXT('My COBOL program') OPTION(*SOURCE) "
```

Use case #10: Simple Java application

This use case involves compiling a simple Java™ application on i5/OS. The Java compiler is directly accessible to the PASE environment shell, so there is no need to use the PASE system command.

The steps for this use case are:

1. Execute the Java compiler, pointing it to the Java source file. For example:

```
javac HelloWorld.java
```

Use case #11: Composite application

This use case involves building an application that has parts whose source code is in Java and other parts whose source code is RPG. Many applications include Web application elements, hosted in IBM® WebSphere®, that provide a front-end for other logic implemented in RPG.

The Web application may be dependent on the RPG interfaces, and it is often desirable to build all elements of the application in a coordinated fashion, so that the change to one element is not promoted until the corresponding change to the other element is also built. This use case is an example of such a scenario.

This use case also demonstrates threaded project configuration. The Java compilation is executed in parallel with the RPG compilation. Threaded project configuration is a more efficient process for producing the composite application, even though all of the steps are executed on the same i5/OS platform.

The steps for this use case are:

2. Execute use case #3 and use case #8 as parallel threaded steps for the same project.
3. Include a post-compile step that blocks additional processing until all compile threads complete.

Trademarks

IBM, the IBM logo, and Rational are trademarks of IBM Corporation in the United States, other countries, or both. Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both. Other company, product, or service names may be trademarks or service marks of others.

Share this article

 [Digg this story](#)

 [Post to del.icio.us](#)

[Slashdot it!](#)