WebSphere® IBM WebSphere Multichannel Bank Transformation Toolkit

IBM

Version 6.1.2

Migration guide from BTT version 4.3 to BTT version 6.1.2

Note!
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 31.

This edition applies to Version 6, Release 1, Modification 2, of *IBM WebSphere Multichannel Bank Transformation Toolkit* (5724-H82) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You can send to the following address:

IBM China Software Development Lab
Bank Transformation Toolkit Product
Diamond Building, ZhongGuanCun Software Park, Dongbeiwang West Road No.8,
ShangDi, Haidian District, Beijing 100193 P. R. China

Include the title and order number of this book, and the page number or topic related to your comment.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Migration guide from BTT version 4.3 to BTT version 6.1.2

This migration guide provided by Bank Transformation Toolkit (BTT) version 6.1.2 helps users of BTT version 4.3 move up to the full power and versatility of IBM® WebSphere® Application Server. It provides the migration path to allow an enterprise to reuse much of its existing toolkit application code base and still take advantage of the WebSphere J2EE environment.

IBM recognizes that an enterprise cannot always upgrade its application software all at once. Because the enterprise can decide how much or how little toolkit functionality to retain, BTT version 6.1.2 enables the enterprise to upgrade to a full J2EE environment incrementally. With BTT version 6.1.2, the enterprise can bypass the toolkit functionality and access J2EE components directly upon requests.

BTT version 6.1.2 provides a migration tool that helps you to migrate your applications. The tool can migrate BTT version 4.3 definitions to BTT version 6.1.2 definitions and migrate BTT version 4.3 class packages and APIs to the corresponding BTT version 6.1.2 class packages and APIs.

This migration guide describes the concepts that you need to know before you migrate your applications to BTT version 6.1.2 and the tasks you need to take to roll out the migrations with the migration tool.

Concepts

Before you can perform the migration tasks, you need to understand the concepts that you might meet during the migration process. In this chapter, you can have an overview of the migration, find out how you can benefit from the migration, and get an understanding of the migration process. For detailed information about concepts in the migration from BTT version 4.3 to BTT version 6.1.2, see the following sections.

Migration Overview

BTT version 6.1.2 provides the following functions to help you migrate your applications from BTT version 4.3 to BTT version 6.1.2:

Definition Transformation
This feature transforms the definition files from BTT version 4.3 to BTT version 6.1.2. All the existing definition files are converted or migrated to the corresponding BTT version 6.1.2 definition files. For example, the definition file *dse.ini* is converted to another definition file *btt.xml*.

Java™ Code Migration
Because many components in BTT version 6.1.2 are different from those in BTT version 4.3, the migration tool in BTT version 6.1.2 provides a function to make the previous code migrated from the existing BTT version 4.3 program to the BTT version 6.1.2 base program. This reduces the workload for the application program migration.

Customer Extension Support
This is a key feature of the migration tool. The migration tool is rule-based, which means that the specific migration action depends entirely on the given rules. You can customize your own migration rules to meet the program migration requirements.

Default Migration Rules Provided
BTT version 6.1.2 provides default migration rules for migration from BTT version 4.3 to BTT version 6.1.2. You can leverage the default rules directly or extend it to meet your additional requirements.

Exclusions
UI features here refer to the JSP presentation logic and Swing based presentation logic. In BTT version 6.1.2, Swing based Java client is totally compatible with BTT version 4.3, so no migration is needed. As for JSP presentation logic, you need to migrate it manually.

Constraints
To reduce the complexity of the migration process, the migration tool limits the availability. The migration tool is rule based, so you are not able to migrate the migration scenarios that rules cannot represent.

Benefits gained through the migration

Bank Transformation Toolkit version 6.1.2 provides support for distributed architecture. The adoption of a distributed architecture brings some immediate benefits to the system planner and developer.

You can benefit from the migration in the following aspects:

- Scalability improves by an order of magnitude. A version 6.1.2 system can scale from a small or medium enterprise having a single server to large enterprises with server farms distributed across multiple regional data centers.
- Performance can be tuned by balancing hardware and network usage against the expected user base. Rational® Application Developer and WebSphere Integration Developer provide many tools and guides for performing this tuning.
- Industry standards such as JCA, EJB, JMS, and Web services replace many previous architectures and facilities in the toolkit. This adds flexibility to expanding enterprise applications using third-party components that are built with the same industry standard architectures.
- To enhance its flexibility, the toolkit maintains much of its previous external design philosophy. It further adds more configurability to this mix by allowing its deployment information to be configured. Scaling and performance tuning are all performed using configuration settings rather than Java code changes.
- Platform flexibility is provided using this configurability as well. Version 6.1.2 can execute using Rational Application Developer for small scale deployments or it can make use of the enhanced features of WebSphere Integration Developer to expand the available options to much larger enterprise deployments.

For more information about the benefits brought by BTT version 6.1.2, see the Solution Architecture Document for BTT version 6.1.2.

Migration tool architecture and support

Bank Transformation Toolkit version 6.1.2 provides a migration tool to help you migrate your toolkit applications developed with pervious versions of the toolkit to the new version 6.1.2 architecture.

A high scalability architecture has been applied to this rule based migration tool. Under this architecture, you can easily extend the toolkit to meet your specific requirements. It is much like a generic migration toolkit, because it can be used in most Java application migration cases besides BTT. Therefore, this toolkit is much better than any other similar migration toolkit in BTT history. The following figure

shows the architecture of this rule based migration tool.

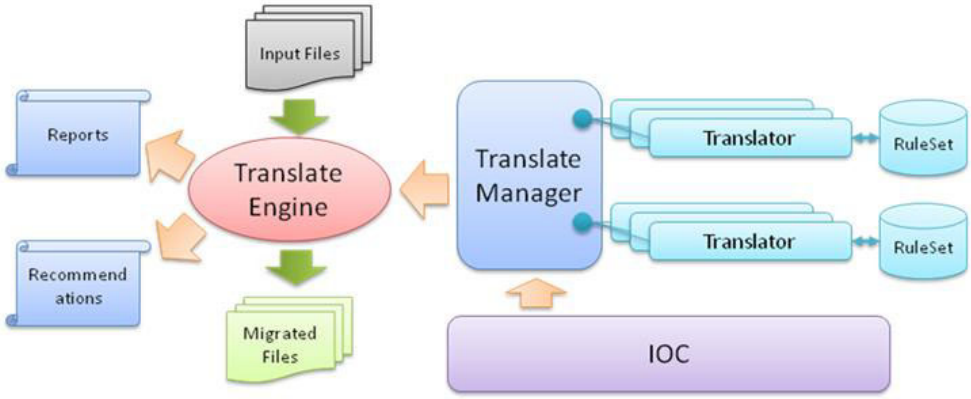


Figure 1. Migration tool architecture

The specific functional descriptions of the migration tool are listed as follows:

- Migrate Java code:

  From BTT version 4.3 to BTT version 6.1.2, there are inevitable API changes to the same functions. It is time-consuming to replace the old API calls with the new ones. Java code migration is one major function of the migration tool.
- Migrate *dse.ini* file:

  Dse.ini contains most of the configurations for all the BTT components. New components are added in version 5.x and version 6.x. In BTT version 6.1.2, the definition file *dse.ini* is converted to another definition file btt.xml. The migration tool helps you to migrate this file.
- Migrate other XML files:

  In BTT, context, formatter, services, data element, and operation are externalized into XML files. The referenced package names and class names of these XML files are different in different versions. Migrating these XML files is another major function of the migration tool.
- Generate migration report:

  After you run the migration tool, a report is generated to record the tasks the migration tool performed.
- Generate migration recommendation

  If some cases cannot be easily migrated by the toolkit, the migration tool gives recommendations on these cases to alert you. Then, you can follow these recommendations to migrate your applications.

The migration tool is rule-based, which means that the migration depends on the given rules. You can change the rule set according to your needs. You can also use this tool to migrate other Java applications.

Migration procedures

Before you start migrating your applications from BTT version 4.3 to BTT version 6.1.2, you need to follow the migration procedures.

The whole migration process consists of the following four phases:

- Phase 1: Analysis and preparation
- Phase 2: Customizing the migration tool

- Phase 3: Migrating your applications
- Phase 4: Adding business logic and fixing errors

The phases are iterative and contain tasks that might be refined during the project life cycle.

Phase 1: Analysis and preparation

In this phase, you need to analyze the differences between BTT version 4.3 and BTT version 6.1.2, especially the application logic layer.

Based on the documentation of the existing application system, you can identify the parts that can be migrated automatically and the parts that cannot be migrated automatically, and then make necessary preparation for the migration. To get well prepared for the migration, you need to do the following work:

1. Gather all the definition files of the existing application system version 4.3. The migration tool can migrate the definition files on the server side from version 4.3 to version 6.1.2. The migration tool can process the definition contents based on the given rules. BTT version 6.1.2 provides default migration rules, but if there are any special tag definitions that are not included in the default rules, the migration tool might not be able to migrate them successfully or properly. In this case, you need to customize the migration tool rules to extend the process to cover those special tag definitions for the application or migrate them manually later.
2. The migration tool can migrate the business logic Java code of the application to BTT version 6.1.2 directly. BTT version 6.1.2 uses the processor, operation, and operation step on the server side of BTT, which is the same as in BTT version 4.3. The differences are in package names, class names, and changes of APIs. The migration tool can migrate BTT version 4.3 classes and APIs to the corresponding BTT version 6.1.2 classes and APIs.
3. BTT version 6.1.2 provides more powerful Context and Formatter, and the API usage is different with that in BTT version 4.3. The migration tool of BTT version 6.1.2 can help you migrate the Context, Formatter, and APIs or provide recommendations for the migration.
4. Do manual migration to run the BTT version 4.3 event in BTT version 6.1.2, because the prior event mechanism is changed to fit BTT version 6.1.2 framework.
5. Migrate BTT version 4.3 services. The supported services of BTT version 6.1.2 on the server side does not support all of the BTT version 4.3 services, and the SNA LU0/6.2 communication services are changed to JCA connectors. You must find a substitutive solution (or implement a new service based on the new service architecture) for these unsupported services. For example, Lotus Notes® support, LDAP support, MQ connector, and the relevant JCA connector. The migration tool can migrate the server side service definitions with customized migration rules, but you must find a substitutive solution for the unsupported services.
6. The event mechanism is changed to fit version 6.1.2 framework so that it can be distributable and spans different servers. You must check and modify the applications accordingly if the event mechanism is adopted in the server side in the current application system.
7. There are two kinds of client applications in BTT version 4.3. There is no change to the Swing based Java client, so you do not need to migrate it to BTT version 6.1.2. For JSP/HTML based client, the migration tool does not support its migration, so manual migration is needed.

8. Migrate the BTT trace. BTT version 6.1.2 enhances the trace features to support more powerful functions and comprehensive APIs. BTT version 6.1.2 trace is completely compatible with BTT version 4.3 trace. If you do not want to use the new trace framework, the old trace can still work in BTT version 6.1.2 runtime container. If you want to use the new trace, use the migration tool to achieve the trace by defining specific migration rules.
9. If the future application only runs on WebSphere Application Server, use Rational Application Developer to migrate the application. If the future application runs on WebSphere Process Server, use WebSphere Integration Developer to migrate the application.

Phase 2: Customizing the migration tool

In this phase, you need to customize the migration tool to extend its functionality.

To customize the migration tool, take the following steps:

1. Customize the Java code migration rules. Because the migration tool is based on migration rules, if you need extend the BTT functionality, you should customize the migration rules to meet your special requirements.
2. Customize the definition file migration rules. Because the migration tool is based on migration rules, if you need to extend the BTT functionality and have some special tags in the definitions, you should customize the migration rules to meet these special requirements.

Phase 3: Migrating your applications

In this phase, you can start migrating your applications with the migration tool.

To migrate your applications from BTT version 4.3 to BTT version 6.1.2, take the following steps:

1. Replace the old BTT version 4.3 jar files with BTT version 6.1.2 jar files, and change the build path of the project to the new jar files.
2. Import the application that you want to migrate to the workspace as a project, and configure the migration tool. The migration tool will migrate the application directly based on the original project.
3. Migrate the configuration file *dse.ini* to *btt.xml*. In BTT version 4.3, the configuration file is *dse.ini*, but in BTT version 6.1.2, it is renamed as *btt.xml*. See the following figure.

Figure 2. Migrating the configuration file

4. Migrate the definition files of the context, the data element, the type data, and the format.
5. Migrate the business logic. BTT version 6.1.2 uses process, operation, and operation step as channel aware business logic, which is similar with BTT version 4.3. In BTT version 6.1.2, the architecture is refined, and some components are re-designed, so the package names, class names, and some APIs in BTT version 6.1.2 are different from those in BTT version 4.3. The migration tool will automatically migrate these differences.
6. Manually migrate the components that cannot be completely migrated by the migration tool, based on the recommendation provided by the migration tool.

Phase 4: Adding the business logic and fixing errors

After you have migrated the applications from BTT version 4.3 to BTT version 6.1.2, you must add the business logic and fix the errors.

To add the business logic and fix errors, take the following steps:

1. Fix the Jave build errors. There might be some API mismatch problems between BTT version 4.3 and BTT version 6.1.2. The migration tool can handle most of them or provide recommendations for manual migration. If not, leverage RAD to fix the problem. For detailed information about API usage, see the related tasks.
2. Build the services that BTT version 6.1.2 does not support, and change the access to the services.
3. Change the access to the communication services to comply with JCA.
4. Do JSP migration manually with the help of RAD. The migration tool does not support JSP migration.

Using the migration tool

The Bank Transformation Toolkit (BTT) provides a migration tool to help you migrate your toolkit applications from version 4.3 to version 6.1.2. The migration tool can perform the following tasks to help you migrate your applications:

- Convert the file *dse.ini* from BTT version 4.3 to the file *btt.xml* in BTT version 6.1.2.
- Migrate the definition files in BTT version 4.3 to the definition files in BTT version 6.1.2, including context definitions, formatter definitions, and so on.
- Migrate the application Java codes, including package names, class names, APIs, and so on from BTT version 4.3 to BTT version 6.1.2.
- Migrate the JSP Tablib uri, BTT customized tag, page import, and Java code in JSP file from BTT version 4.3 to BTT version 6.1.2.

The migration tool is rule-based. It migrates BTT version 4.3 applications to BTT version 6.1.2 based on the given migration rules. You can customize the migration rules to meet your special requirements.

The migration tool is extendable. It provides three extension points, by which you can implement your own migration rule engine to meet your special migration requirements.

For information about how to use the migration tool, see Migration Tool.

Tasks

Bank Transformation Toolkit version 6.1.2 provides a migration tool to help you migrate your toolkit applications that are developed with BTT version 4.3 to the new version 6.1.2 architecture.

After you get to know the concepts in the migration process, you can use the migration tools to migrate your applications from BTT versoin 4.3 to BTT version 6.1.2. In this chapter, you can find information about how to set up the migration tool and how to use the migration tool to perform the migration tasks.

Setting up the migration tool

Before you can migrate your applications from BTT version 5.2 to BTT version 6.1.2, you need to set up the migration tool.

If you have installed the toolkit before you install IBM Rational Application Developer on your workstation, you need to copy the following plug-in files to the $D(RAD)/plugins directory manually after you install IBM Rational Application Developer:

- **com.ibm.btt.tools.migration_6.1.2**
- **com.ibm.btt.core_6.1.2**

To set up the migration tool, take the following steps:

1. Start IBM Rational Application Developer⌂
2. Import the project that you want to migrate.
3. Click **Window** → **Preferences** on the menu bar of IBM Rational Application Developer.
4. In the dialog box that is displayed, select **BTT Migration** in the left panel.
5. In the right panel of the dialog box, select the migration rule definition file that you want to apply. In the migration tool plug-in, there is a **config** category, and you can find BTT version 5.2 to BTT version 6.1.2 migration rule definition files under the related category.
6. Click **OK**.
7. In the **Project Explorer** view, right-click the BTT definition file and select **BTT Migration**.

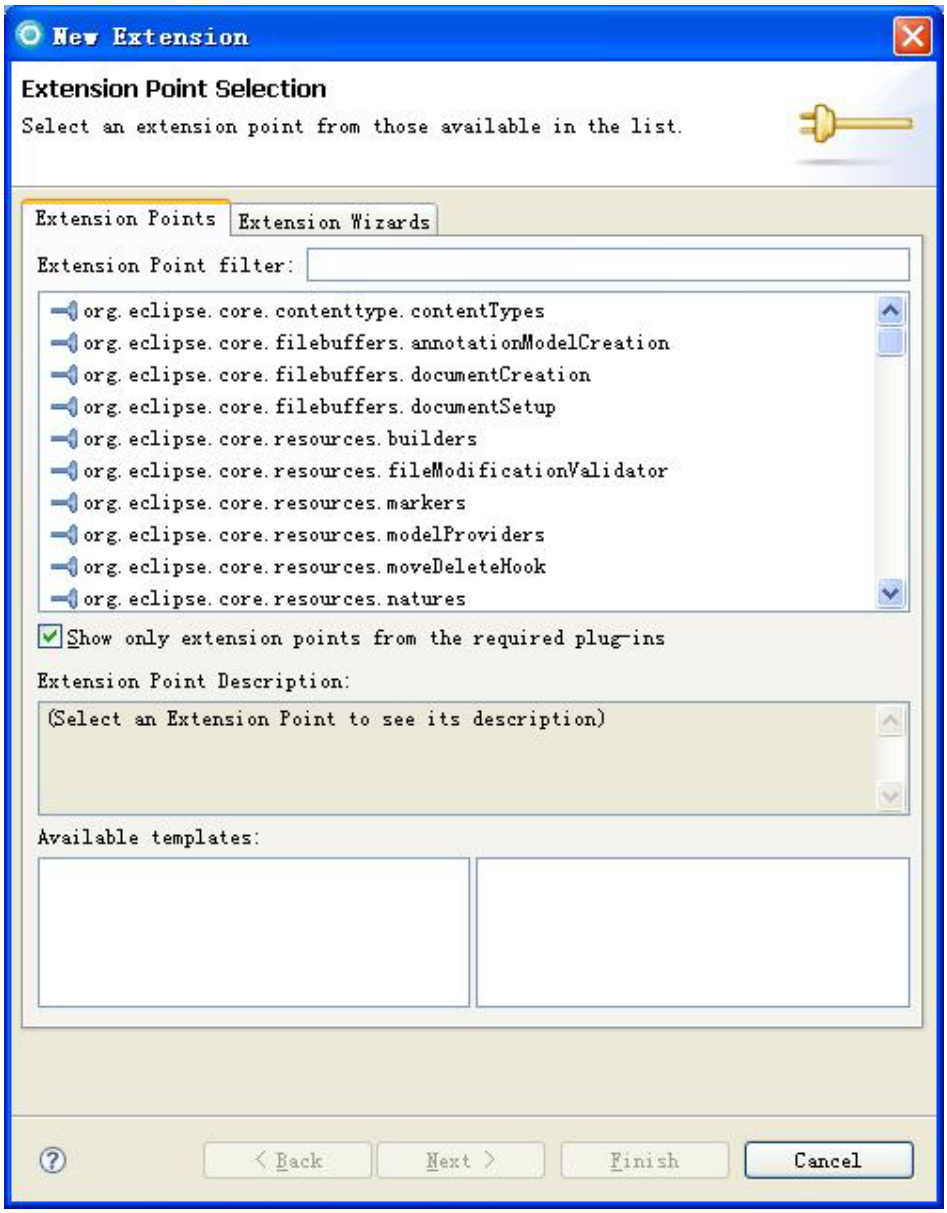
The migration report and migration results will be displayed in the **Project Explorer** view if there are any.

Adding Translators

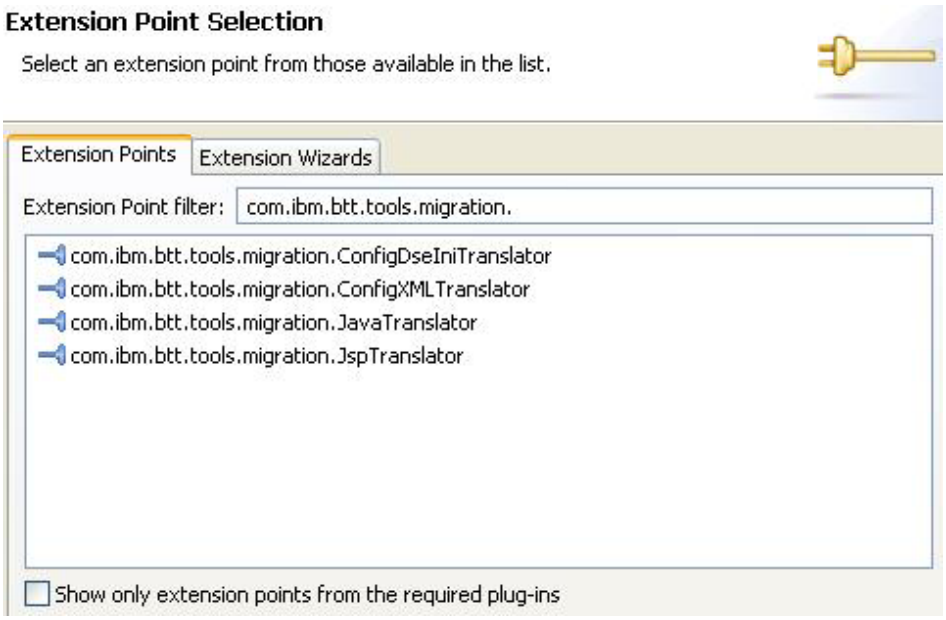
You can extend the Migration Tool and add your own Translators in your plug-in project.

To add a Translator, do the following:

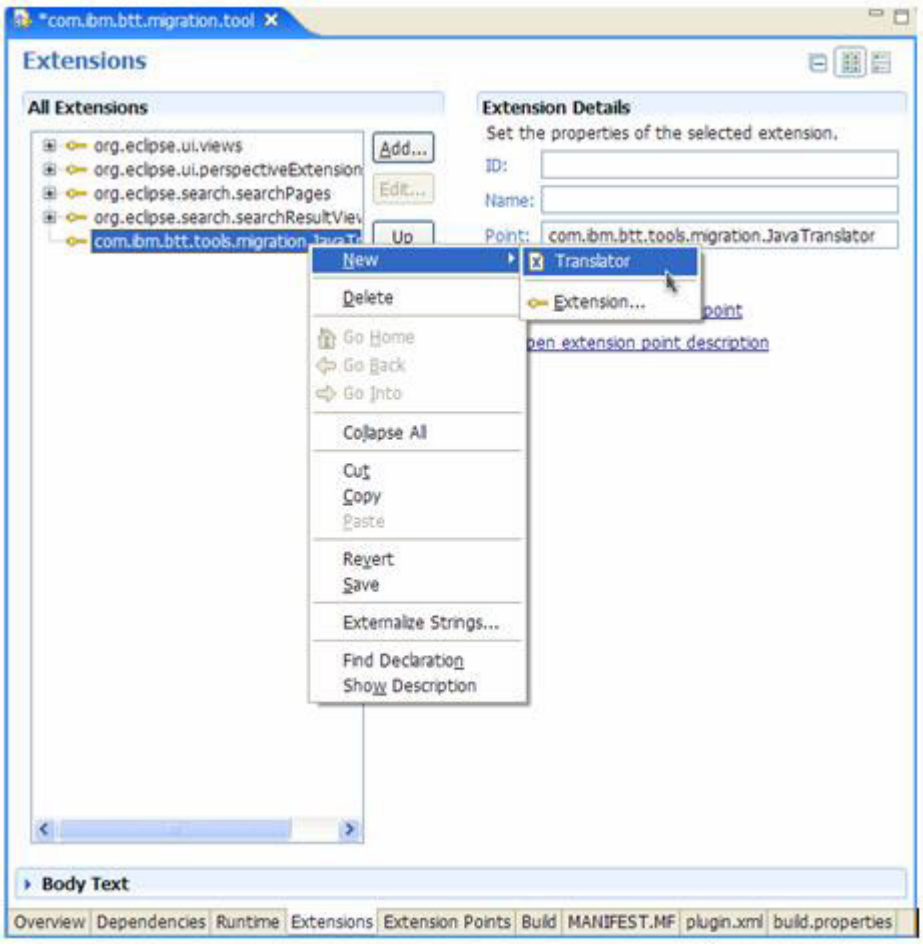
- Create a Plug-in project:
  - In the Rational Application Developer, click **File** → **New** → **Others...**. Expand **Plug-in Development** and select **Plug-in Project**, and click **Next**.
  - Enter the name of the project, and click **Next**, and then in the dialog box that pops up, click **Finish**.
  - The Plug-in project is created.
- Add new extensions:
  - Click the **Extensions** tab, and click **Add...**.
  - The **New Extension** dialog box pops up:



- Clear the **Show only extension points from the required plug-ins** checkbox.
- There are four extension points available: `ConfigDseIniTranslator`, `ConfigXMLTranslator`, `JavaTranslator`, and `JspTranslator`. Select the extension points, and click **Finish** to add these extensions:

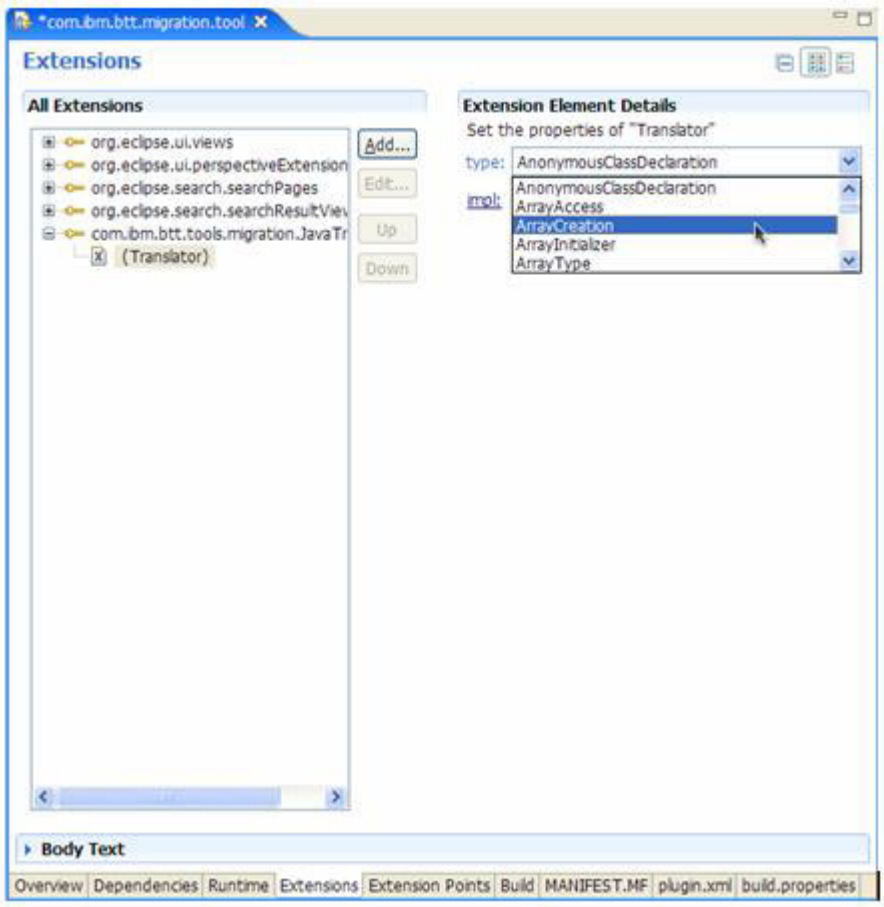


- Create the extension instance. Take JavaTranslator for example:
  - Right-click `com.ibm.btt.tools.migration.JavaTranslator`, and select **New** → **Translator**.



  - The extension instance is created.

- In the right panel **Extension Element Details**, click the **type** dropdown list, and select the extension type and then click **Browse...** to implement the corresponding Translator interface.



Adding rules

The migration tool provides four types of migration: Java migration, BTT definition XML file migration, dse.ini file migration, and JSP file migration. The migration is implemented according to the configured migration rules.

Two series of rule sets are provided: version 4.3 to version 6.1.2 migration rule set and version 5.2 to version 6.1.2 migration rule set. The rule sets are in the $D(RAD)\plugins\com.ibm.btt.tools.migration_6.1.2\config directory. By default, the 4.3 to 6.1.2 rule set is applied.

There are three configuration files for this Rule-based Migration Tool:

- javaRule.xml: rules for Java code migration
- cfgRule.xml: rules for XML and dse.ini file migration
- bttTemplate.xml: btt.xml template.
- jspRule.xml: rules for JSP file migration

If the default rules can not meet your migration needs, you can add new rules and configure them as the migration rules in the Preference setting.

To configure your own migration rules, do the following:

- Click **Window** → **Preferences...** in Rational Application Developer.
- In the left panel of the Preferences dialog box, expand **BTT**, and select **Migration**.
- The migration configuration is displayed in the right panel.
- You have two options for the configuration of the migration rule:
  - Default rule: BTT migration tool provides a default migration rule. If the default rule meets your migration requirements, you can use it directly. If you do not specify any migration rule, the default rule will be used.
  - Self-define rule: If the default rule does not meet your migration requirements, you can define your own migration rule, and then click **Browse** to locate your own migration file.

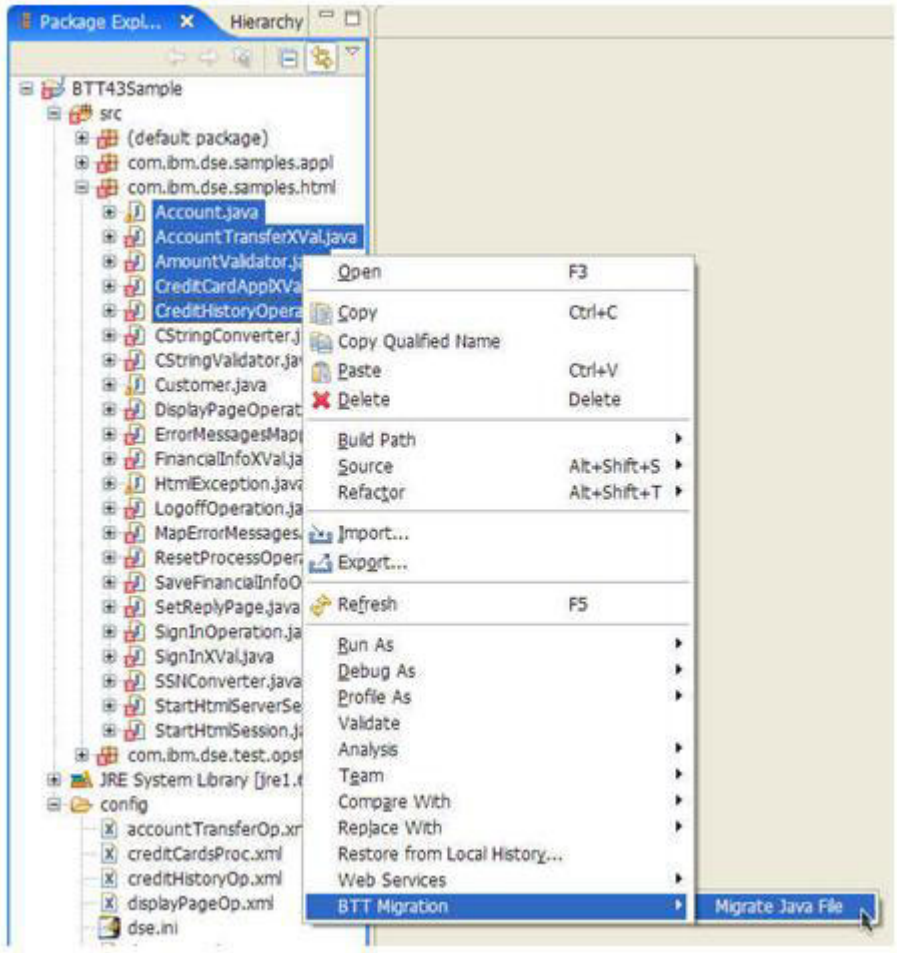
The migration tool reports error if the migration rule is configured incorrectly or the rule file is not valid.

Migrating Java code

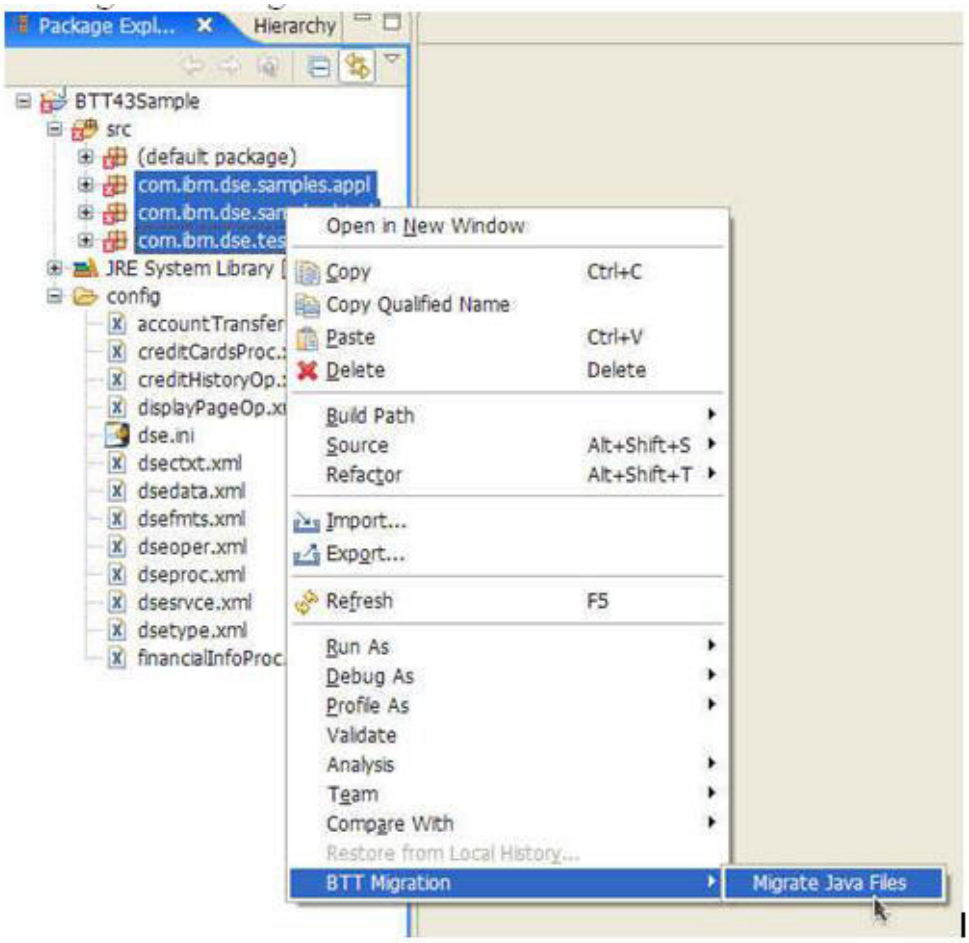
Java code migration can be done in two ways: Java file level migration and package level migration.

To migrate Java code, do the following:

- Select the target Java file or package, and right-click it.
- If you are doing Java file level migration, in the menu that pops up, click **BTT Migration** → **Migrate Java File**:



If you are doing package level migration, click **BTT Migration → Migrate Java Files**:



- The Java code is migrated. The migrated Java code will replace the original Java code.

Migrating dse.ini file

To migrate dse.ini file, do the following:

- Select the dse.ini file and right-click it.
- In the menu that pops up, click **BTT Migration** → **Migrate dse.in**.
- The dse.ini file is migrated into the config.migrated folder.

Migrating XML file

To migrate XML file, do the following:

- Select the XML file and right-click it.
- In the menu that pops up, click **BTT Migration** → **Migrate XML file**.
- The XML file is migrated into the config.migrated folder.

Batch migration is also supported.

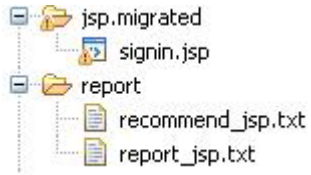
Migrating JSP file

You can migrate JSP Tablib uri, BTT customized tag, page import, and Java code in JSP file. The migration rules of taglib uri and BTT customized tag are defined in the JSP migration rule file. The migration rules of Java code are defined in Java migration rule file.

To migrate JSP file, perform the following steps:

1. Select the JSP file and right-click it.
2. In the menu that pops up, click **BTT Migration → Migrate JSP file** as shown in the following screen capture.

3. The JSP file is then migrated to the jsp.migrated folder.



4. During the migration, all the activities are recorded into the report_jsp.txt file. The tool also provides migration recommendations in the recommand_jsp.txt file. You can use these two files to judge if you need to do any further migration manually. All the report files and recommendation files are generated in a folder named **report**.

The tool supports multiple JSP files migration. You can migrate multiple JSP files at one time. Use one of the following ways to do the multiple JSP files migration:

1. Select all the JSP files you want to migrate, and right-click them and then select **BTT Migration → Migrate JSP file**. Then all the JSP files are migrated.
2. Select a folder which contains multiple JSP files that you want to migrate, and right-click the folder and then select **BTT Migration → Migrate JSP files**. Then all the JSP files in this folder are migrated.

Following are the samples for each migration type:

Table 1. Taglib uri migration

Before migration	Migration rule	After migration
<%@ taglib uri="/WEB-INF/dse.tld" prefix="dse" %>	<taglibRule oldTagUrl="/WEB-INF/dse.tld" newTagUrl="/WEB-INF/btt.tld" oldTagPrefix="dse" newTagPrefix="btt"/>	<%@ taglib uri="/WEB-INF/btt.tld" prefix="btt" %>

Table 2. Tag migration

Before migration	Migration rule	After migration
<dse:label text="jspMigrationToolTest"/>	<tagRule oldTitle="dse:label" newTitle="btt:label" oldKey="text" newKey="text_new" oldVal="jspMigrationToolTest" newVal= "jspMigrationToolTest_new"/>	<btt:label text_new="jsp MigrationToolTest_new"/>

Table 3. Page import migration

Before migration	Migration rule	After migration
<%@page import="com.ibm.dse" %>	<pageImport oldPattern="com.ibm.dse" newPattern="com.ibm.btt"/>	<%@page import="com.ibm.btt"%>

Table 4. Java code migration

Before migration	Migration rule	After migration
com.ibm.dse.base. JavaExtensions.getAlphaUniqueCode()	<simpleRule oldItem="*com.ibm.dse*" newItem="*com.ibm.btt*" />	com.ibm.btt.base. JavaExtensions. getAlphaUniqueCode()

Migrating externalizer

To migrate externalizer, you need to migrate the `getSetting()` parameter and the iniFile path.

- Migrating the `getSetting()` parameter:

  In BTT version 6.1.2, each component has its own initializer. So you need to use the component's initializer to get the parameter.

  Following is the code sample before migration:

  ```
  String startupOpName = (String) Settings.getSettings().getValueAt("channelHandlers." +
   channelContext.getDeviceType() + ".startupOp");
  ```

  After you migrate to BTT version 6.1.2, the code is as follows:

  ```
  String startupOpName = (String) ChannelInitializer.getSettings().
  getValueAt(channelContext.getDeviceType() + ".startupOp");
  ```

- Migrating iniFile path:

  BTT version 6.1.2 supports two kinds of iniFile: one is jar path iniFile, such as `jar:///package/dse.ini`; the other is file path iniFile, such as `file:///path/dse.ini`. If you use the file path way, you need to migrate the iniFile in `StartServerServlet` class:

  ```
  String res=null;
  if (!iniFileParameter.startsWith("/WEB-INF")) {
  File iniFile=new File(iniFileParameter);
  if (iniFile.isAbsolute() && iniFile.exists()) {
  res=iniFile.getAbsolutePath();
  }
  //TODO BTT611, use file path way initialize file name.
  res = "file:\\\\\\" + res;
  }
  ```

Migrating Session Management

This topic introduces how you can migrate BTT version 4.3 Session Management to BTT version 6.1.2 Session Management.

BTT provides the following utility method in the *com.ibm.btt.sm.CSSessionHandler* class:

Public static void *markSessionExpired(HttpSession aSession)* throws BTTSMException. After this method is called, the session will be marked as expired and will be removed at the end of operation. You need to call this method in the logoff operation. Then, the session context will be cleaned and the HTTP session object will be destroyed after you log off the application.

BTT version 4.3 supports both `cookie=true` and `cookie=false` modes.

- When `cookie=true`, HttpSessionHandler object is used in the HttpSession that implements the interface `HttpSessionBindingListener`. When the HttpSession is invalidated, the `endSession(sessionId)` method is triggered.
- When `cookie=false`, BTT channel is responsible for the session expiration. It uses a dedicated thread CSSessionManager to check the expired sessions regularly with the method `checkExpiredSessions()`. And it ends the expired BTT session.

In BTT version 6.1.2, only `cookie=true` mode is supported. Session management is delegated to HttpSession. You need to add the HttpSession listener in your WAR project for session management.

To migrate the `cookie=false` mode in BTT version 4.3, you need to migrate your old mechanism to the HttpSession timeout mechanism. Add the following HttpSession timeout listener in the *web.xml* file:

```
<listener>
<listener-class>com.ibm.btt.sm.TimeoutHandler</listener-class>
</listener>
```

Migrating Channel

The channel component is restructured in BTT version 6.1.2. Besides `ChannelRequest` and `ChannelResponse`, which help to achieve protocol and channel independency, there are some other changes in structure and code, including:

- In BTT version 4.3, `CSReqServlet` implements `ChannelDriver`. In BTT version 6.1.2, `CSReqServlet` is only a servlet and does not implement `ChannelDriver` interface.
- BTT version 6.1 introduces a `BTTChannelDriver` class, which implements ChannelDriver interface.
- Because `BTTChannelDriver` implements `ChannelDriver` in BTT version 6.1.2. Some functions and methods in `CSReqServlet` are not in `ChannelDriver`.

To migrate channel code, you only need to move some methods and logic to the classes in BTT version 6.1.2.

Migrating Trace

This topic introduces how you can migrate BTT version 4.3 trace to BTT version 6.1.2 trace.

The original trace API of BTT version 4.3 can be still used in BTT version 6.1.2. BTT version 6.1.2 trace facility can automatically map the old API to the new API of BTT version 6.1.2 trace facility. BTT version 6.1.2 traces by package, but the original trace is by the Component ID in BTT version 4.3. BTT version 4.3 trace can trace to multiple targets at the same time, but BTT version 6.1.2 supports tracing to one target at the same time only when the BTTLogFactoryImplementClass is set.

The implementation of trace to File is changed to using Java util API, so the original parameter of trace to File is not supported in BTT version 6.1.2. The original function and implementation of trace to Display is kept in BTT version 6.1.2, so the parameter for trace to Display is still supported.

The original HML trace level is not supported in BTT version 6.1.2. The original trace type is mapped to a new trace level. See the following table.

Table 5. Trace level map

BTT 6.1.2 trace level	BTT 4.3 trace type	WAS trace level	Common logging	Log4J
FATAL	Severe	Fatal	Fatal	Fatal
ERROR	Error	Severe	Error	Error
WARN	Warning	Warning	Warn	Warn
INFO	Information Display	Info	Info	Info
DEBUG	Debug AllTypes	Detail*	Debug	Debug

An example of migrating BTT version 4.3 trace to BTT version 6.1.2 trace is as follows:

BTT version 4.3 trace configuration

```
<kColl id="traces">
    <field id="initializer" value="com.ibm.btt.base.TraceInitializer" />
    <field id="traceToFile" value="yes" />
    <field id="traceToDisplay" value="yes" />
    <field id="traceToWAS" value="yes" />
    <field id="traceWindowTitle" value="Server Trace" />
    <field id="showOriginator" value="yes" />
    <field id="showWarningMessage" value="no" />
    <field id="traceLevels" value="debug" />
    <field id="traceFileName" value="C:\dse\dselog.txt" />
    <field id="traceMaxLogFiles" value="5" />
    <field id="font" value="monospaced" />
    <field id="createBackup" value="yes" />
    <field id="fileNumberOfLines" value="4000" />
    <field id="displayNumberOfLines" value="2000" />
    <field id="linesOfBuffer" value="7000" />
    <field id="lineLength" value="200" />

    <kColl id="requestersComponents">
      <traceRequester id="#CHA" trace="yes" traceLevels="HML" traceTypes="FATAL" />
      <traceRequester id="#CS" trace="yes" traceLevels="HML" traceTypes="DEBUG" />
    </kColl>
</kColl>
```

BTT version 6.1.2 trace configuration after migration

```
<kColl id="traces">
    <field id="initializer" value="com.ibm.btt.base.TraceInitializer" />
    <field id="traceTargetFactoryImplClass" value="com.ibm.btt.base.BTTLogFactoryToDisplayImp" />
```

```
    <field id="displayNumberOfLines" value="2000" />
    <kColl id="requestersComponents">
      <traceRequester id="com.ibm.btt.base.*" trace="yes" traceLevels="FATAL" />
      <traceRequester id="com.ibm.btt.channel.*" trace="yes" traceLevels="DEBUG" />
    </kColl>
</kColl>
```

BTT version 4.3 trace application code

```
if (Trace.doTrace(Constants.CHACOMPID,Trace.High,Trace.Debug))
    Trace.trace(Constants.CHACOMPID,Trace.High,Trace.Debug,Settings.getTID(),
                "CHA Debug .........");
if (Trace.doTrace(Constants.CHACOMPID,Trace.High,Trace.Information))
    Trace.trace(Constants.CHACOMPID,Trace.High,Trace.Information,Settings.getTID(),
                "CHA info .........");
```

BTT version 6.1.2 trace application code after migration

```
BTTLog   log=BTTLogFactory.getLog("com.ibm.btt.base.LocalContextImp");
If (log.doDebug())
  log.debug("CHA Debug......");
If (log.doInfo())
  log.info("CHA info......");
```

Four trace types

BTT version 6.1.2 supports four trace target types: trace to window, trace to self define file, trace to WAS file, trace to common-logging.

- Trace to window and self-defined file:

  Trace to window and trace to self-defined file functions are kept for migration consideration.

  Trace to window function can be used at client side.

  Most of the functions of trace to file are not supported in BTT version 6.1.2. For example, the following is not supported:

  ```
  <field id="traceMaxLogFiles" value="5"/>
  <field id="fileNumberOfLines" value="4000"/>
  <field id="displayNumberOfLines" value="200"/>
  <field id="linesOfBuffer" value="700"/>
  <field id="lineLength" value="128"/>
  ```

  As a result, you can use trace to common logging (for example, Log4J) or trace to WAS instead.

- Trace to WAS and common logging:

  BTT version 6.1.2 supports trace to WAS and common logging. BTT trace maps the BTT trace API to the corresponding trace API of common logging and WAS java.util.logging.

  Note: Common logging only provides the API standard, and the logging implementation is provided by other logging facilities for example Log4J. The trace level is also converted.

  1. Trace to WAS:
     a. Change dse.ini file, and enable traceToWAS. You can enable and disable trace switch and set trace level for each component.
     b. You can enable and change the trace level dynamically in WAS console.
     c. Trace application code calls the BTT trace API. You can input component ID and trace level as parameters.
     d. BTT trace facility calls the corresponding java.util.logging API, if you enable the component and BTT trace level. The BTT trace level is converted to WAS trace level.

e. After BTT trace calls the WAS trace API, and if the input trace level is enabled in the trace configuration of WAS console, the trace content is recorded to WAS trace file.

2. Trace to common logging:
   a. Change the dse.ini file and enable traceToCommonLogging.
   b. You can enable or disable the trace switch and set the trace level for each component.
   c. Set the common logging property file and specify the implementation of common logging, for exmaple Log4J.
   d. Trace application code calls the corresponding common logging API. You can input component ID and trace level as parameters.
   e. BTT trace facility calls the corresponding common-logging API, if you enable the component and the BTT trace level. The BTT trace level is converted to common logging trace level.
   f. BTT trace calls the common logging API. Whether the trace content is recorded into the trace file depends on the trace level setting of the common logging implementation.
3. Migrating code:
   – Migrating XML definition files.

     Before migration, the code for XML definition file is as follows:

```
<kColl id="traces">
  <field id="traceToFile" value="yes"/>
  <field id="traceToDisplay" value="yes"/>
  <field id="traceFileName" value="c:\dse\log\btt.log"/>
  <field id="traceMaxLogFiles" value="100"/>
  <field id="font" value="monospaced"/>
  <field id="createBackup" value="yes"/>
  <field id="fileNumberOfLines" value="4000"/>
  <field id="displayNumberOfLines" value="200"/>
  <field id="linesOfBuffer" value="700"/>
  <field id="lineLength" value="128"/>
  <field id="showOriginator" value="yes"/>
  <field id="useServletsEngineLog" value="no"/>
  <field id="servletsEngineLogPort" value="80"/>
  <field id="showWarningMessage" value="yes"/>
  <field id="traceWindowTitle" value="PSP6 Server Trace"/>
  <field id="traceTypes" value="DIPEWSV"/>
  <field id="traceLevels" value="HML"/>
  <field id="showContextDump" value="yes"/>
  <kColl id="requestersComponents">
  ....
  </kColl>
</kColl>
```

     After migration, the code is as follows:

```
<kColl id="traces">
  <field id="initializer" value="com.ibm.btt.base.TraceInitializer"/>
  <field id="traceTargetFactoryImplClass"
   value="com.ibm.btt.base.BTTLogFactoryToCommonLoggingImp" />
  <field id="showContextDump" value="yes"/>
  ....
</kColl>
```

     In the XML trace definition migration, `<requestersComponents>` definition is not used for common logging. You need to configure `initializer` and `traceTargetFactoryImplClass`. You can leave the other configuration as it is.

     If you choose common logging trace, you need to migrate the `TraceConfigServlet` class too.

If you choose WAS trace, WAS console has similar functions.

– Some trace APIs do not need migration, for example:

```
Trace.trace(CommonsConstants.COMPID, Trace.Low, Trace.Information, Settings.getTID(), msg);
Trace level=Trace.Low, Setting.getTID() will not use,
BTT trace will map the other three parameter to commond-logging.
```

Migrationg exception handling

In BTT version 4.3, APIs such as `context.getKeyedCollection()`, `context.getValue()` do not throw out exceptions; while in BTT version 6.1.2, these APIs throw exceptions.

As a result, you need to migrate exception handling and you must trace the exceptions.

The following code is the code sample before migration:

```
return utb.getContext().getKeyedCollection().getElements().entrySet();
```

After migration, the code is as follows:

```
//TODO , btt612, exception handling
try {
return utb.getContext().getKeyedCollection().getElements().entrySet();
} catch (DSEInvalidRequestException e) {
// TODO: Exception handling
//Trace the exception.
}
return null;
```

Migrating APIs

To migrate APIs, do the following changes:

- Change `Settings.reset(iniFile)` to `InitManager.reset(iniFile)`
- Use `Context ctx=(Context)ContextFactory.createContext(name, false)` to create context, where `False` refers to local context.
- Change `SessionEntry. getTimeStamp()` to `CSSessionHandler.getTimeStamp(SessionEntry)`.
- Change `(HttpServletRequest)channelContext.getChannelRequest()` to `(HttpServletRequest)channelContext.getChannelRequest().getRequest()`.
- Change `HttpServletResponse res=(HttpServletResponse)channelContext.getChannelResponse().getResponse()` to `(HttpServletRequest)channelContext. getChannelResponse ().getResponse ()`.
- Change `Context.getRoot()` to `ContextFactory.getRoot()`.
- Change `Context.getSession(sessionId)` to `CSSessionHandler.getSession(sessionId)`.
- Change `Context.getCurrentContextForSession(sessionId)` to `CSSessionHandler.getCurrentContextForSession(sessionId)`.
- Change `Context.getTIDForSession(s)` to `CSSessionHandler.getTIDForSession(string)`.
- Change `Context.setSessionObjectForSession()` to `CSSessionHandler. setSessionObjectForSession()`.
- Change `Context.setTimeStampForSession()` to `CSSessionHandler.setTimeStampForSession()`.

- Change `Context.removeSession(sessionId)` to `CSSessionHandler.removeSession(sessionId)`
- Change `Context.getTypeForSession()` to `CSSessionHandler.getTypeForSession()`.
- Change `Context.getSessionTable()` to `CSSessionHandler.getSession(sessionId)`
- Change `formatter.unformat(Label13Value,Operation)` to `formatter.unformat(Label13Value,Context)`.
- Change `Formatter.format(Operation)` to `Formatter.format(Context)`.
- In the HtmlRequestHandler, change return type to `Object` in `public ServerOperation executeRequest(ChannelContext channelContext)`.

Migrating J2EESecurityService

BTT version 4.3 has the J2EESecurityService, which is not included in BTT version 6.1.0. To use J2EESecurityService in BTT version 6.1.2, you can develop specific code for it.

Migrating the XML files

To migrate the XML file, take the following steps:

1. Right-click the XML file.
2. In the menu that is displayed, select **BTT Migration → Migrate XML file**. Then the XML file is migrated to the **config.migrated** folder.

Note: Batch migration is also supported for the migration of XML files.

Migrating context

This topic introduces how you can migrate BTT version 4.3 Context to BTT version 6.1.2 Context.

You do not need to change the context, type, and data definitions of BTT version 4.3 when you migrate them to BTT version 6.1.2, because these definitions are the same in BTT version 4.3 and BTT version 6.1.2. In BTT version 6.1.2, a class table tag for context implementation class and an initializer tag are added in the CHA configuration in the BTT XML definition file, such as the file *btt.xml*.

The APIs in BTT version 4.3 and BTT version 6.1.2 are almost the same except the way of constructing the context instance. In BTT version 6.1.2, the context is constructed from ContextFactory. The service reference of BTT version 4.3 is not useful in BTT version 6.1.2. An example of migrating BTT version 4.3 context to BTT version 6.1.2 context is as follows:

1. Creating BTT version 4.3 context.

```
Context ctxt = (Context) Context.readObject("myContext");
Context ctxt2 = new com.ibm.dse.base.Context();
ctxt2.setName("myContext2");
```

2. Creating BTT version 6.1.2 context.

```
Context diiTestCtx = ContextFactory.createContext("myContext");
Context ctxt2 = new com.ibm.btt.base.Context();
ctxt2.setName("myContext2");
```

Migrating formatter

You need to migrate the configuration that is related to the Formatter in the file *dse.ini* in BTT version 4.3 to the file *btt.xml* in BTT version 6.1.2 and keep all the other code the same as before.

Migrating event

This topic introduces how you can migrate BTT version 4.3 Event to BTT version 6.1.2 Event.

You can migrate BTT version 4.3 Event to BTT version 6.1.2 Event by changing the package name in the operation definition file. See the following example:

```
<operation id="eventManagerServerOperation"
    implClass="com.ibm.btt.event.EventManagerServerOperation">
<refFormat name="csReplyFormat" refId="EventManagerCSFormat" />
</operation>
```

Migrating Session Management

This topic introduces how you can migrate BTT version 4.3 Session Management to BTT version 6.1.2 Session Management.

BTT provides the following utility method in the *com.ibm.btt.sm.CSSessionHandler* class:

Public static void *markSessionExpired(HttpSession aSession)* throws BTTSMException. After this method is called, the session will be marked as expired and will be removed at the end of operation. You need to call this method in the logoff operation. Then, the session context will be cleaned and the HTTP session object will be destroyed after you log off the application.

BTT version 4.3 supports both `cookie=true` and `cookie=false` modes.

- When `cookie=true`, HttpSessionHandler object is used in the HttpSession that implements the interface `HttpSessionBindingListener`. When the HttpSession is invalidated, the `endSession(sessionId)` method is triggered.
- When `cookie=false`, BTT channel is responsible for the session expiration. It uses a dedicated thread CSSessionManager to check the expired sessions regularly with the method `checkExpiredSessions()`. And it ends the expired BTT session.

In BTT version 6.1.2, only `cookie=true` mode is supported. Session management is delegated to HttpSession. You need to add the HttpSession listener in your WAR project for session management.

To migrate the `cookie=false` mode in BTT version 4.3, you need to migrate your old mechanism to the HttpSession timeout mechanism. Add the following HttpSession timeout listener in the *web.xml* file:

```
<listener>
<listener-class>com.ibm.btt.sm.TimeoutHandler</listener-class>
</listener>
```

Migrating Trace

This topic introduces how you can migrate BTT version 4.3 trace to BTT version 6.1.2 trace.

The original trace API of BTT version 4.3 can be still used in BTT version 6.1.2. BTT version 6.1.2 trace facility can automatically map the old API to the new API of BTT version 6.1.2 trace facility. BTT version 6.1.2 traces by package, but the original trace is by the Component ID in BTT version 4.3. BTT version 4.3 trace can trace to multiple targets at the same time, but BTT version 6.1.2 supports tracing to one target at the same time only when the BTTLogFactoryImplementClass is set.

The implementation of trace to File is changed to using Java util API, so the original parameter of trace to File is not supported in BTT version 6.1.2. The original function and implementation of trace to Display is kept in BTT version 6.1.2, so the parameter for trace to Display is still supported.

The original HML trace level is not supported in BTT version 6.1.2. The original trace type is mapped to a new trace level. See the following table.

Table 6. Trace level map

BTT 6.1.2 trace level	BTT 4.3 trace type	WAS trace level	Common logging	Log4J
FATAL	Severe	Fatal	Fatal	Fatal
ERROR	Error	Severe	Error	Error
WARN	Warning	Warning	Warn	Warn
INFO	Information Display	Info	Info	Info
DEBUG	Debug AllTypes	Detail*	Debug	Debug

An example of migrating BTT version 4.3 trace to BTT version 6.1.2 trace is as follows:

BTT version 4.3 trace configuration

```
<kColl id="traces">
    <field id="initializer" value="com.ibm.btt.base.TraceInitializer" />
    <field id="traceToFile" value="yes" />
    <field id="traceToDisplay" value="yes" />
    <field id="traceToWAS" value="yes" />
    <field id="traceWindowTitle" value="Server Trace" />
    <field id="showOriginator" value="yes" />
    <field id="showWarningMessage" value="no" />
    <field id="traceLevels" value="debug" />
    <field id="traceFileName" value="C:\dse\dselog.txt" />
    <field id="traceMaxLogFiles" value="5" />
    <field id="font" value="monospaced" />
    <field id="createBackup" value="yes" />
    <field id="fileNumberOfLines" value="4000" />
    <field id="displayNumberOfLines" value="2000" />
    <field id="linesOfBuffer" value="7000" />
    <field id="lineLength" value="200" />

    <kColl id="requestersComponents">
      <traceRequester id="#CHA" trace="yes" traceLevels="HML" traceTypes="FATAL" />
      <traceRequester id="#CS" trace="yes" traceLevels="HML" traceTypes="DEBUG" />
    </kColl>
</kColl>
```

BTT version 6.1.2 trace configuration after migration

```
<kColl id="traces">
    <field id="initializer" value="com.ibm.btt.base.TraceInitializer" />
    <field id="traceTargetFactoryImplClass" value="com.ibm.btt.base.BTTLogFactoryToDisplayImp" />
    <field id="displayNumberOfLines" value="2000" />
    <kColl id="requestersComponents">
      <traceRequester id="com.ibm.btt.base.*" trace="yes" traceLevels="FATAL" />
      <traceRequester id="com.ibm.btt.channel.*" trace="yes" traceLevels="DEBUG" />
    </kColl>
</kColl>
```

BTT version 4.3 trace application code

```
if (Trace.doTrace(Constants.CHACOMPID,Trace.High,Trace.Debug))
    Trace.trace(Constants.CHACOMPID,Trace.High,Trace.Debug,Settings.getTID(),
                "CHA Debug .........");
if (Trace.doTrace(Constants.CHACOMPID,Trace.High,Trace.Information))
    Trace.trace(Constants.CHACOMPID,Trace.High,Trace.Information,Settings.getTID(),
                "CHA info .........");
```

BTT version 6.1.2 trace application code after migration

```
BTTLog   log=BTTLogFactory.getLog("com.ibm.btt.base.LocalContextImp");
If (log.doDebug())
  log.debug("CHA Debug......");
If (log.doInfo())
  log.info("CHA info......");
```

Four trace types

BTT version 6.1.2 supports four trace target types: trace to window, trace to self define file, trace to WAS file, trace to common-logging.

- Trace to window and self-defined file:

  Trace to window and trace to self-defined file functions are kept for migration consideration.

  Trace to window function can be used at client side.

  Most of the functions of trace to file are not supported in BTT version 6.1.2. For example, the following is not supported:

  ```
  <field id="traceMaxLogFiles" value="5"/>
  <field id="fileNumberOfLines" value="4000"/>
  <field id="displayNumberOfLines" value="200"/>
  <field id="linesOfBuffer" value="700"/>
  <field id="lineLength" value="128"/>
  ```

  As a result, you can use trace to common logging (for example, Log4J) or trace to WAS instead.

- Trace to WAS and common logging:

  BTT version 6.1.2 supports trace to WAS and common logging. BTT trace maps the BTT trace API to the corresponding trace API of common logging and WAS java.util.logging.

  Note: Common logging only provides the API standard, and the logging implementation is provided by other logging facilities for example Log4J. The trace level is also converted.

  1. Trace to WAS:
     a. Change dse.ini file, and enable traceToWAS. You can enable and disable trace switch and set trace level for each component.
     b. You can enable and change the trace level dynamically in WAS console.

c. Trace application code calls the BTT trace API. You can input component ID and trace level as parameters.

d. BTT trace facility calls the corresponding java.util.logging API, if you enable the component and BTT trace level. The BTT trace level is converted to WAS trace level.

e. After BTT trace calls the WAS trace API, and if the input trace level is enabled in the trace configuration of WAS console, the trace content is recorded to WAS trace file.

2. Trace to common logging:

a. Change the dse.ini file and enable traceToCommonLogging.

b. You can enable or disable the trace switch and set the trace level for each component.

c. Set the common logging property file and specify the implementation of common logging, for exmaple Log4J.

d. Trace application code calls the corresponding common logging API. You can input component ID and trace level as parameters.

e. BTT trace facility calls the corresponding common-logging API, if you enable the component and the BTT trace level. The BTT trace level is converted to common logging trace level.

f. BTT trace calls the common logging API. Whether the trace content is recorded into the trace file depends on the trace level setting of the common logging implementation.

3. Migrating code:

– Migrating XML definition files.

Before migration, the code for XML definition file is as follows:

```
<kColl id="traces">
  <field id="traceToFile" value="yes"/>
  <field id="traceToDisplay" value="yes"/>
  <field id="traceFileName" value="c:\dse\log\btt.log"/>
  <field id="traceMaxLogFiles" value="100"/>
  <field id="font" value="monospaced"/>
  <field id="createBackup" value="yes"/>
  <field id="fileNumberOfLines" value="4000"/>
  <field id="displayNumberOfLines" value="200"/>
  <field id="linesOfBuffer" value="700"/>
  <field id="lineLength" value="128"/>
  <field id="showOriginator" value="yes"/>
  <field id="useServletsEngineLog" value="no"/>
  <field id="servletsEngineLogPort" value="80"/>
  <field id="showWarningMessage" value="yes"/>
  <field id="traceWindowTitle" value="PSP6 Server Trace"/>
  <field id="traceTypes" value="DIPEWSV"/>
  <field id="traceLevels" value="HML"/>
  <field id="showContextDump" value="yes"/>
  <kColl id="requestersComponents">
  ....
  </kColl>
</kColl>
```

After migration, the code is as follows:

```
<kColl id="traces">
  <field id="initializer" value="com.ibm.btt.base.TraceInitializer"/>
  <field id="traceTargetFactoryImplClass"
   value="com.ibm.btt.base.BTTLogFactoryToCommonLoggingImp" />
  <field id="showContextDump" value="yes"/>
  ....
</kColl>
```

In the XML trace definition migration, `<requestersComponents>` definition is not used for common logging. You need to configure `initializer` and `traceTargetFactoryImplClass`. You can leave the other configuration as it is.

If you choose common logging trace, you need to migrate the `TraceConfigServlet` class too.

If you choose WAS trace, WAS console has similar functions.

– Some trace APIs do not need migration, for example:

```
Trace.trace(CommonsConstants.COMPID, Trace.Low, Trace.Information, Settings.getTID(), msg);
Trace level=Trace.Low, Setting.getTID() will not use,
BTT trace will map the other three parameter to commond-logging.
```

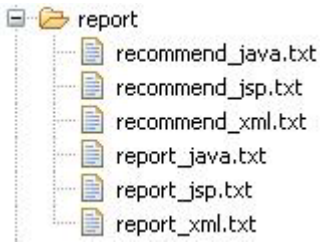
Generating reports and recommendations

After the migration, the reports are generated to record what has been done during the migration process. There are separate reports for Java code migration and BTT XML definition file migration.

Note: The report and recommendation files will not be generated in dse.ini migration.

The recommendations for some special cases are also generated to warn and guide you about how to do further migration manually.

All the report files and recommendation files are generated in a folder named report.



References

For the references related to migration from BTT version 4.3 to BTT version 6.1.2, see the following topics in this chapter.

Manual migration

The migration tool cannot migrate everything from BTT version 4.3 to BTT version 6.1.2 automatically. You must do some manual migration, including:

- Some new features in BTT version 6.1.2, such as Web 2.0, Rich Client, and so on. The base functionality of the migration tool only migrates the existing applications to BTT version 6.1.2 and makes sure that they can run in BTT version 6.1.2. The new features in BTT version 6.1.2 are not migrated by the migration tool, so you must re-design the applications and implement them manually.
- The existing services that are out of the scope of BTT version 6.1.2. These services in BTT version 4.3 application system are not included in the migration tool, so you need to migrate them manually.

- The server side event mechanism of BTT version 6.1.2 is changed. You can define event related migration rules in BTT version 6.1.2, but for some complicated cases that migration rules cannot handle, you must migrate them manually.
- The communication service access uses JCA connector in BTT version 6.1.2. You must modify it manually.

Diagnosis for the migration tool

After each migration task, the migration tool will save the migration messages into a report file. The file is stored in the project workspace under the category named **report**. The recommendations for manual migration are saved in the recommendation file under the same category. In this file, the parts that need manual migration are listed out, and recommendations for manual migration are provided. You can perform the necessary manual migration according to the recommendations.

Code changes in dse.ini after migration

This section lists the code changes in dse.ini after you migrate your application from BTT version 4.3 to version 6.1.2.

You can use the migration tool to migrate dse.ini file, and after migration, the file name is changed from dse.ini to btt.xml.

Context

In BTT version 4.3, all the parameters for context are defined under `<kColl id="context">` in des.ini, as shown in the following code sample:

```
<kColl id="contexts">
     <field id="context" value="com.ibm.dse.base.Context"/>
    </kColl>
```

After you migrate to BTT version 6.1.2, the definition of Context and CHA contains client side CHA and server side CHA in btt.xml.

Following is the code sample:

```
<!-- ====================================== -->
  <!--               Context                -->
  <!-- ====================================== -->
  <!-- Context is object defines and encapsulates a set of resources -->
  <!-- (data and services) for functional or business organization entity. -->
  <kColl id="Context" initializer="com.ibm.btt.base.ContextInitializer">
   <field id="extFile" value="context.xml" />
   <field id="initializer" value="com.ibm.btt.base.ContextInitializer"/>
   <kColl id="classTable">
    <field id="context" value="com.ibm.btt.base.LocalContextImpl"/>
   </kColl>
   <!-- Release below configuraton items, otherwise, keep them close -->
  <!--field id="ejbInitialContextFactory"
         value="com.ibm.websphere.naming.WsnInitialContextFactory"/-->
   <!--field id="EJBProviderURL" value="iiop://remoteserverip:remoteserverport"/-->
   <!-- Indicates the home interface of CHAEJB -->
   <!--field id="CHASessionLocalHome" value="java:comp/env/ejb/CHASession"/-->
   <!--field id="CHAInstanceLocalHome" value="java:comp/env/ejb/CHAInstance"/-->
   <!--field id="CHAControlLocalHome" value="java:comp/env/ejb/CHAControl"/-->
   <!--field id="CHASessionHome" value="ejb/com/ibm/btt/cha/ejb/CHASessionHome61"/-->
   <!--field id="CHAInstanceHome" value="ejb/com/ibm/btt/cha/ejb/CHAInstanceHome61"/-->
   <!-- Indicates the initialized context tree of CHA-->
   <!--field id="initTailContextName" value="branchServer"/-->
```

```
   <!-- Indicates whether to cleanup the context instance from the DB tables-->
   <!--field id="cleanupCHAServer" value="true"/-->
   <!--field id="isLocalCall" value="true"/-->
   <!--field id="isPersistenceEnabled" value="false"/-->

  </kColl>
```

Initializer and extFile

In BTT version 6.1.2, the configuration of each component is defined in the component's kColl. And each component has its own initializer.

In the following example, you can see that after the migration, each data's definition kColl has extFile and initializer:

Before migration, the code is as follows:

```
<kColl id="data">
  <field id="field" value="com.ibm.btt.base.DataField"/>
  <field id="iColl" value="com.fortis.be.rbaa.commons.data.IndexedCollection"
   description="compound"/>
  <field id="kColl" value="com.ibm.btt.base.KeyedCollection" description="compound"/>
  <field id="operDef" value="com.ibm.btt.base.OperField"/>
  <field id="refData"/>
  <field id="sessionEntry" value="com.ibm.btt.base.SessionEntry"/>
  <field id="sessionTable" value="com.ibm.btt.base.SessionTable"/>
  <!-- Fortis WSBCC Commons -->
  <field id="finalField" value="com.fortis.be.commons.data.FinalDataField"/>
</kColl>
```

After migration, the code is changed to the following:

```
<kColl id="data">
<field id="extFile" value="dsedata.xml" />
<field id="initializer"
value="com.ibm.btt.base.DataInitializer" />
  <kColl id="classTable">
    <field id="field" value="com.ibm.btt.base.DataField"/>
    <field id="iColl" value="com.fortis.be.rbaa.commons.data.IndexedCollection"
   description="compound"/>
    <field id="kColl" value="com.ibm.btt.base.KeyedCollection" description="compound"/>
    <field id="operDef" value="com.ibm.btt.base.OperField"/>
    <field id="refData"/>
    <field id="sessionEntry" value="com.ibm.btt.base.SessionEntry"/>
    <field id="sessionTable" value="com.ibm.btt.base.SessionTable"/>
    <!-- Fortis WSBCC Commons -->
    <field id="finalField" value="com.fortis.be.commons.data.FinalDataField"/>
  </kColl>
</kColl>
```

Processor definition

In BTT version 4.3, the processor definition is as follows:

```
<kColl id="processors">
    <procDef id=".." value=".." path=".."/>
    ....
</kColl>
```

After you migrate to BTT version 6.1.2, the processor definition is changed to the following:

```
<kColl id="processors">
    <kColl id="files">
        <procDef id=".." value=".." path=".."/>
```

```
        ....
    </kColl>
    ....
</kColl>
```

Operation definition

In BTT version 4.3, the operation definition is as follows:

```
<kColl id="operation">
  <procDef id=".." value=".." path=".."/>
  ....
</kColl>
```

After you migrate to BTT version 6.1.2, the operation definition is changed to the following:

```
<kColl id="operation">
  <kColl id="files">
    <operDef id=".." value=".." path=".."/>
    ....
  </kColl>
  ....
</kColl>
```

Package

In BTT version 4.3, the component package is under its own kColl:

```
<kColl id="packages">
    <kColl id="operations">
        <field id="packageEJS1" value="com.ibm.dse.appl.ej.server"/>
    </kColl>
    <kColl id="processors">
        <field id="package1" value="com.ibm.dse.automaton"/>
        <field id="package2" value="com.ibm.dse.automaton.ext"/>
    </kColl>
    <kColl id="typedData">
        <field id="package1" value="com.ibm.dse.base.types.ext"/>
        <field id="package2" value="com.ibm.dse.base.types"/>
    </kColl>
</kColl>
```

After you migrate to BTT version 6.1.2, all the packages for type, operation and processor are under the `<kColl id="packages">`. Following is the code sample:

```
<kColl id="type">
    <kColl id="packages">
        <field id="package1" value="com.ibm.dse.base.types.ext"/>
        <field id="package2" value="com.ibm.dse.base.types"/>
    </kColl>
    ....
</kColl>
```

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM China Software Development Lab
Diamond Building, ZhongGuanCun Software Park, Dongbeiwang West Road No.8, ShangDi, Haidian District, Beijing 100193 P. R. China

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java is a trademark of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.