



创建应用程序以处理可变长度和多段输出消息

目录

为包含可变长度和多段的 IMS 事务创建

J2C 应用程序 1

创建应用程序以处理可变长度和多段输出消息. . . . 1

课程 1.1: 选择资源适配器 2

课程 1.2: 设置 Web 项目以及 Java 接口和实现. . . 5

课程 1.3: 创建消息缓冲区类 6

课程 1.4: 创建 Java 测试类以测试应用程序. . . . 11

总结: 为包含可变长度和多个段的 IMS 事务创建 J2C
应用程序 14

为包含可变长度和多段的 IMS 事务创建 J2C 应用程序

本教程描述如何使用 J2C Java™ Bean 向导构建简单的 Web 应用程序来处理带有输入和输出数据（包含可变长度和多段）的 IMS™ 事务。

学习目标

本教程使您能够完成下列任务：

- 使用 J2C Java Bean 向导来创建运行 IMS 事务的 J2C Java bean。
- 创建消息缓冲区类，并使用 doclet 注释来编辑此类。
- 为 J2C Java bean 创建方法以运行 IMS 事务并为方法提供输入和输出数据类型。
- 为输出消息段创建 Java 数据绑定。
- 创建测试 Java 类以测试应用程序

所需时间

30 分钟

相关信息



[查看 PDF 版本](#)

[多段输出样本](#)

创建应用程序以处理可变长度和多段输出消息

本教程将指导您完成一些详细步骤来生成一个 J2C 应用程序以处理可变长度和多段 IMS 事务输出消息。

本教程可能需要一些可选可安装组件。要确保您已安装了适当的可选组件，请参阅“系统要求”列表。

本教程分为一些练习，必须按顺序完成这些练习，教程才有意义。本教程教您如何使用 J2C Java bean 向导来创建运行 IMS 事务的 Java bean。在完成这些练习的过程中，您将执行下列操作：

- 使用 J2C Java Bean 向导来创建运行 IMS 事务的 J2C Java bean。
- 创建消息缓冲区类 CCIBuffer.java，并使用 doclet 注释来编辑此类。
- 为 J2C Java bean 创建方法以运行 IMS 事务并为方法提供输入和输出数据类型。
- 为输出消息段创建 Java 数据绑定。
- 创建测试 Java 类 TestMultiSeg.java 以调用运行 IMS 事务的 J2C Java bean 方法，然后用 IMS 事务返回的数据填充缓冲区中的各个输出段。

注：测试 Java 类 *TestMultiSeg.java* 是为英语语言环境创建的。对于其他语言环境，可能需要修改代码。

所需时间

完成本教程大约需要 30 分钟。如果研究其他与本教程相关的概念，则完成本教程可能需要更长时间。

技能级别

熟练

用户

本教程面向熟悉企业信息系统（EIS），特别是熟悉 IMS 的用户。

系统要求

为了完成本教程，需要安装下列工具和组件：

- IBM® WebSphere® Application Server V6.1
- J2EE 连接器（J2C）工具
- 与 **IMS** 环境有关的信息：在本教程中，您的应用程序与 IMS 中的 IMS 应用程序进行交互。需要获取一些信息，例如，IMS Connect 的主机名和端口号以及将运行事务的 IMS 数据存储器的名称。请与 IMS 系统管理员联系以获取此信息。特别地，如果要运行 IMS 程序 IMS\MultiSegmentOutput，则需要在 IMS 中执行一些设置工作。
- **COBOL** 文件 **MSOut.cbl** 的一个副本：可在以下产品安装目录中找到此文件：\rad\eclipse\plugins\com.ibm.j2c.cheatsheet.content_7.0.0\samples\IMS\MultiSegmentOutput。如果想将它存储到本地，可从以下链接复制其代码：第 3 页的『MSOut.cbl』
- 一个干净的工作空间。

要使用本教程，必须安装并配置了应用程序服务器。要验证是否提供了服务器运行时环境，请单击**窗口** → **首选项**，展开**服务器**，然后单击**已安装的运行时**。可以使用此窗格来添加、除去或编辑已安装的服务器运行时定义。还可以下载并安装对新服务器的支持。

必备知识

为了能够彻底完成本教程，您应该熟悉下列内容：

- J2EE 和 Java 编程
- 基本 IMS Transaction Manager（IMS TM）概念

课程 1.1：选择资源适配器

本课程指导您完成一些详细的步骤来选择并配置资源适配器以连接至 IMS 服务器。

本教程中使用的 IMS 事务不是一个 IMS 安装验证程序。本教程使用了 DFSDDLTO，这是一个根据控制语句信息发出对 IMS 的调用的 IMS 应用程序。以下提供了用于本教程的 DFSDDLTO 控制语句。但是，要运行本教程，您必须为 DFSDDLTO 配置您的环境并提供必需的 JCL。本教程将 SKS2 用作此 DFSDDLTO 应用程序的事务代码。

DFSDDLTO 控制语句

```
S11 1 1 1 1    TP      1
L      GU
E      OK
E  Z0017 DATA  SKS2 M2 SI1M3 SI1
WTO SEGMENT SI1 RECEIVED
L      GN
E      QD
WTO END OF INPUT SEGMENTS
L      ISRT IW06OUT
L  Z0012 DATA  *****M1S01
E      OK
WTO SEGMENT S01 INSERTTED
L      ISRT
L  Z0027 DATA  *****M1S02*****M2S02
E      OK
```

2 创建应用程序以处理可变长度和多段输出消息

```

WTO SEGMENT S02 INSERTTED
L      ISRT
L  Z0048 DATA  *****M1S03*****M2S03*****M3S03
E      OK
WTO SEGMENT S03 INSERTTED
WTO CURRENT PROGRAM STLDDLT2 TERMINATED
L      GU

```


本教程使用了 COBOL 数据结构来描述 IMS 事务输入和输出消息。注意 IMS 返回的输出消息由固定长度的三段构成:

- OUTPUT-SEG1 (16 字节)
- OUTPUT-SEG2 (31 字节)
- OUTPUT-SEG3 (52 字节)

此特定 IMS 应用程序返回的输出消息固定大小为 99 个字节, 并通过 COBOL 01 结构 OUTPUT-MSG 表示。

开发此多段应用程序的一个方法是使用 COBOL 定义 OUTPUT-MSG 来定义此事务的输出。第二个方法是为此事务的输出创建一个输出消息。本教程提供的代码使用了第二个方法, 因为这个方法也可用于构建处理可变长度输出消息的应用程序。将继续使用每个消息段的 COBOL 定义, 以简化对每个段的数据访问。

连接至 IMS 服务器

1. 如果工作空间的右上角选项卡中未出现 J2EE 图标  , 则需要切换至 J2EE 透视图。从菜单栏中, 选择 **窗口 > 打开透视图 > 其他**。将打开“选择透视图”窗口。
2. 选择 **J2EE**。
3. 单击**确定**。将打开 J2EE 透视图。
4. 在 J2EE 透视图, 选择 **文件 > 新建 > 其他**。
5. 在“新建”页面中, 选择 **J2C > J2C Java Bean**。单击**下一步**。
6. 在“选择资源适配器”页面中, 选择 J2C 1.5 IMS 资源适配器。对于本教程, 请选择 **IMS Connector for Java (IBM: 9.1.0.2.3)**。单击**下一步**。
7. 在“连接属性”页面中, 清除**受管连接**并选择**非受管连接**。(在本教程中, 将使用非受管连接来直接访问 IMS。)接受缺省的连接类名 `com.ibm.connector2.ims.ico.IMSManagedConnectionFactory`。在空白字段中, 输入所有必需的连接信息。用星号 (*) 指示的必填字段有:
 - a. 对于 **TCP/IP 连接**:
 - 1) **主机名**: (必填) IMS Connect 的 IP 地址或主机名。
 - 2) **端口号**: (必填) 目标 IMS Connect 使用的端口号。
 - b. 对于**本地选项连接**:
 - 1) **IMS Connect 名称**: (必填) 目标 IMS Connect 的名称。
 - 2)
 - c. 对于**两者**:
 - 1) **数据存储器名称**: (必填) 目标 IMS 数据存储器的名称。
8. 可从您的 IMS 系统管理员处获取连接信息。当提供了必需的连接信息后, 单击**下一步**。

MSout.cbl

以下是 MSout.cbl 中的代码:

MSout.cbl

IDENTIFICATION DIVISION.
program-id. pgml.
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

DATA DIVISION.

```
*
*      IMS TOC Connector for Java, Multi-segment Output Example
*
*****/
*                                          */
* (c) Copyright IBM Corp. 1998                      */
*   All Rights Reserved                               */
*   Licensed Materials - Property of IBM              */
*
*                                          */
*   DISCLAIMER OF WARRANTIES.                      */
*
*   The following (enclosed) code is provided to you solely for the */
*   purpose of assisting you in the development of your applications. */
* The code is provided "AS IS." IBM MAKES NO WARRANTIES, EXPRESS OR */
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF */
*   MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING */
*   THE FUNCTION OR PERFORMANCE OF THIS CODE.
* IBM shall not be liable for any damages arising out of your use */
* of the generated code, even if they have been advised of the */
* possibility of such damages.
*
*                                          */
*   DISTRIBUTION.                                  */
*
*   This generated code can be freely distributed, copied, altered, */
*   and incorporated into other software, provided that:
*   - It bears the above Copyright notice and DISCLAIMER intact
*   - The software is not for resale
*
*                                          */
*****/
*
```

LINKAGE SECTION.

01 INPUT-MSG.

```
02 IN-LL      PICTURE S9(3) COMP.
02 IN-ZZ      PICTURE S9(3) COMP.
02 IN-TRCD    PICTURE X(5).
02 IN-DATA1   PICTURE X(6).
02 IN-DATA2   PICTURE X(6).
```

01 OUTPUT-MSG.

```
02 OUT-ALLSEGS PICTURE X(99) VALUE SPACES.
```

01 OUTPUT-SEG1.

```
02 OUT-LL      PICTURE S9(3) COMP VALUE +0.
02 OUT-ZZ      PICTURE S9(3) COMP VALUE +0.
02 OUT-DATA1   PICTURE X(12) VALUE SPACES.
```

01 OUTPUT-SEG2.

```
02 OUT-LL      PICTURE S9(3) COMP VALUE +0.
02 OUT-ZZ      PICTURE S9(3) COMP VALUE +0.
02 OUT-DATA1   PICTURE X(13) VALUE SPACES.
02 OUT-DATA2   PICTURE X(14) VALUE SPACES.
```

01 OUTPUT-SEG3.

```
02 OUT-LL      PICTURE S9(3) COMP VALUE +0.
```


02	OUT-ZZ	PICTURE S9(3) COMP VALUE +0.
02	OUT-DATA1	PICTURE X(15) VALUE SPACES.
02	OUT-DATA2	PICTURE X(16) VALUE SPACES.
02	OUT-DATA3	PICTURE X(17) VALUE SPACES.

PROCEDURE DIVISION.

课程 1.2: 设置 Web 项目以及 Java 接口和实现

课程 1.2 将指导您完成创建 Web 项目以及 Java 接口和实现

在开始之前，必须先完成课程 1.1。在本课程中，您将完成下列任务：

- 创建 J2C Java bean
 - 创建动态 Web 项目
1. 在工作台中完成的所有工作都必须与一个项目相关联。项目为工作文件和目录提供了一个有组织的视图，并根据项目的类型对视图功能进行了优化。在工作台中，所有文件都必须位于一个项目中，因此在创建 J2C Java bean 之前，首先需要创建放置此 bean 的项目。在“新建 J2C Java Bean”页面的项目名称字段中，输入值 MultiSegOutput。
 2. 单击项目名称字段旁边的新建按钮以创建新的项目。
 3. 现在，您将创建动态 Web 项目。在“创建新的源项目”页面中，选择 **Web 项目**，并单击下一步。
 4. 在“新建动态 Web 项目”页面中，单击显示高级。
 5. 确保选择了下列值：
 - a. 项目名称: MultiSegOutput
 - b. 项目内容: 接受缺省值
 - c. 目标运行时: WebSphere Application Server V6.1
 - d. 配置: 接受缺省值
 - e. 将项目添加至 **EAR**: 已选中
 - f. **EAR** 项目名称: MultiSegOutputEAR
 6. 单击下一步。
 7. 在“项目构面”页面上，接受缺省值。
 8. 单击下一步。
 9. 在“Web 模块”页面上，接受缺省值。
 10. 单击完成。
 11. 可能会出现一个对话框询问是否要切换到动态 Web 透视图。单击是。
 12. 在“J2C Java Bean 输出属性”页面上：
 - a. 在包名字段中，单击浏览并选择 MultiSegOutput 项目。单击确定。
 - b. 在包名字段中输入 sample.ims。
 - c. 在接口名称字段中输入 MS0。
 - d. 在实现名称字段中输入 MS0Impl。
 - e. 不要选中将会话另存为 **Ant** 脚本。
 13. 单击完成。

课程 1.3: 创建消息缓冲区类

课程 1.3 指导您完成创建消息缓冲区类。

在开始之前，必须完成“课程 1.2: 设置 Web 项目以及 Java 接口和实现”。在本课程中，您将完成下列任务：

- 创建消息缓冲区类
 - 使用 doclet 注释编辑消息缓冲区类
 - 创建输入和输出绑定操作
 - 创建输出段数据映射
1. 首先，将创建消息缓冲区类：依次展开 **MultiSegOutput** 项目、**Java** 资源和**JavaSource**。
 2. 右键单击 **sample.ims** 包，并选择**新建 > 类**以启动“新建类”向导。
 3. 输入 **CCIBuffer** 作为类的名称。接受所有缺省设置。
 4. 单击**完成**。**CCIBuffer** 类将在 Java 编辑器中打开。
 5. 在 **CCIBuffer** 类的注释部分，输入标记 **@type-descriptor.message-buffer**。
 6. 按 **CTRL-S** 组合键以保存更改。注意，在 第 9 页的『CCIBuffer.java』中会自动生成新代码。
 7. 接下来，将创建用于运行 **IMS** 事务的方法以及输入消息数据类型：在“项目资源管理器”中，右键单击 **MSOImpl.java**，然后选择**源 > 将方法添加至 J2C Java bean**。
 8. 在“新建 Java 方法”页面中，单击**添加**。
 9. 输入 **runMultiSegOutput** 作为 Java 方法名称。单击**下一步**。
 10. 单击**新建**以定义输入类型。
 11. 选择 **COBOL_to_Java** 映射。单击**浏览**。
 12. 找到 COBOL 文件 **MSOut.cbl**。单击**打开**。
 13. 单击**下一步**。
 14. 在“COBOL 导入器”页面中，单击**显示高级**。
 - a. 选择下列选项：

表 1. COBOL 导入器参数设置

参数	值
平台名称	Z/OS
代码页	IBM-037
浮点格式名称	IBM 十六进制
外部十进制符号	EBCDIC
字节存储次序名称	大尾数法
远程整数字节存储次序名称	大尾数法
带引号的名称	DOUBLE
Trunc 名称	STD
Nsymbol 名称	DBCS

- b. 单击**查询**来装入数据。
 - c. 显示了一列数据结构。在**数据结构**字段中选择 **INPUT-MSG**。
 - d. 单击**下一步**。
 - e. 不要选中**将会话另存为 Ant 脚本**。
15. 在“保存属性”页面中，缺省类名是 **INPUTMSG**。使用 **InputMsg** 覆盖“类名”。单击**完成**。
- 6** 创建应用程序以处理可变长度和多段输出消息

16. 接下来，将创建输出消息数据类型：单击浏览以定义输出类型。
17. 在“选择数据类型”字段中输入 CC，**CCIBuffer** 将出现在“匹配类型”字段中。选择 **CCIBuffer** 作为输出类型。单击完成。
18. 在“Java 方法”页面中，单击完成。
19. 在“Java 方法”页面中，确保 **interactionVerb** 设置为 **SYNC_SEND_RECEIVE (1)** 以指示与 IMS 的交互涉及到发送后进行接收的交互。
20. 单击完成。
21. 接下来将创建输出段数据映射。首先将创建 **OutputSeg1.java** 类：要完成此步骤，需要使用一个独立的数据映射向导，以便可以只创建数据映射文件。
22. 选择文件 > 新建 > 其他 > **CICS/IMS Java 数据绑定**以调用“数据绑定”向导。
23. 单击下一步。
24. 在选择映射列表中选择 **COBOL_To_Java**。单击浏览以查找 COBOL 副本 **MSOut.cbl**。
25. 单击下一步。
26. 在“COBOL 导入器”页面中，单击显示高级。
 - a. 选择下列选项：

表 2. COBOL 导入器参数设置

参数	值
平台名称	Z/OS
代码页	IBM-037
浮点格式名称	IBM 十六进制
外部十进制符号	EBCDIC
字节存储次序名称	大尾数法
远程整数字节存储次序名称	大尾数法
带引号的名称	DOUBLE
Trunc 名称	STD
Nsymbol 名称	DBCS

- b. 单击**查询**来装入数据。
 - c. 显示了一列数据结构。在**数据结构**字段中选择 **OUTPUT-SEG1**。
 - d. 单击下一步。
27. 在“保存属性”向导中，单击浏览，并选择您先前创建的 **MultiSegOutput** 项目。
28. 单击浏览以选择包名： **sample.ims.data**。
29. 将 Java 类名从 **OUTPUTSEG1** 更改为 **OutputSeg1**。
30. 单击完成。
31. 现在您将创建 **OutputSeg2.java** 类：选择文件 > 新建 > 其他 > **CICS/IMS Java 数据绑定**以调用“数据绑定”向导。
32. 单击下一步。
33. 在选择映射列表中选择 **COBOL_To_Java**。单击浏览以查找 COBOL 副本 **MSOut.cbl**。
34. 在“COBOL 导入器”页面中，单击显示高级。

- a. 选择下列选项:

表 3. COBOL 导入器参数设置

参数	值
平台名称	Z/OS
代码页	IBM-037
浮点格式名称	IBM 十六进制
外部十进制符号	EBCDIC
字节存储次序名称	大尾数法
远程整数字节存储次序名称	大尾数法
带引号的名称	DOUBLE
Trunc 名称	STD
Nsymbol 名称	DBCS

- b. 单击**查询**来装入数据。
 - c. 显示了一列数据结构。在**数据结构**字段中选择 **OUTPUT-SEG2**。
 - d. 单击**下一步**。
35. 在“保存属性”向导中，单击**浏览**来选择您先前创建的 **MultiSegOutput** 项目。
 36. 单击**浏览**以选择包名: **sample.ims.data**。
 37. 将 Java 类名从 **OUTPUTSEG2** 更改为 **OutputSeg2**。
 38. 单击**完成**。
 39. 现在您将创建 **OutputSeg3.java** 类: 选择文件 > 新建 > 其他 > **CICS/IMS Java** 数据绑定以调用“数据绑定”向导。
 40. 单击**下一步**。
 41. 在**选择映射列表**中选择 **COBOL_To_Java**。单击**浏览**以查找 COBOL 副本 **MSOut.cbl**。
 42. 在“COBOL 导入器”页面中，单击**显示高级**。

- a. 选择下列选项:

表 4. COBOL 导入器参数设置

参数	值
平台名称	Z/OS
代码页	037
浮点格式名称	IBM 390 十六进制
外部十进制符号	EBCDIC
字节存储次序名称	大尾数法
远程整数字节存储次序名称	大尾数法
带引号的名称	DOUBLE
Trunc 名称	STD
Nsymbol 名称	DBCS

- b. 单击**查询**来装入数据。
- c. 显示了一列数据结构。在**数据结构**字段中选择 **OUTPUT-SEG3**。
- d. 单击**下一步**。

43. 在“保存属性”向导中，单击**浏览**来选择您先前创建的 **MultiSegOutput** 项目。
44. 单击**浏览**以选择包名: **sample.ims.data**。
45. 将 Java 类名从 **OUTPUTSEG3** 更改为 **OutputSeg3**。
46. 单击**完成**。

CCIBuffer.java

以下是 CCIBuffer.java 类中生成的代码:

CCIBuffer.java

```
/*
 * Created on Oct 13, 2004
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package sample.ims;

/**
 * @author ivyho
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 * @type-descriptor.message-buffer
 */
public class CCIBuffer implements javax.resource.cci.Record,
    javax.resource.cci.Streamable, com.ibm.etools.marshall.RecordBytes {

    private byte[] buffer_ = null;

    /**
     * @generated
     */
    public CCIBuffer() {
        return;
    }

    /**
     * @generated
     * @see javax.resource.cci.Record#getRecordShortDescription()
     */
    public String getRecordShortDescription() {
        return (this.getClass().getName());
    }

    /**
     * @generated
     * @see javax.resource.cci.Record#hashCode()
     */
    public int hashCode() {
        return (super.hashCode());
    }

    /**
     * @generated
     * @see javax.resource.cci.Streamable#write(OutputStream)
     */
    public void write(java.io.OutputStream outputStream)
        throws java.io.IOException {
        outputStream.write(buffer_);
    }

    /**
```

```

    * @generated
    * @see javax.resource.cci.Record#setRecordShortDescription(String)
    */
    public void setRecordShortDescription(String shortDescription) {
        return;
    }

    /**
     * @generated
     */
    public int getSize() {
        if (buffer_ != null)
            return (buffer_.length);
        else
            return (0);
    }

    /**
     * @generated
     * @see java.lang.Object#toString
     */
    public String toString() {
        StringBuffer sb = new StringBuffer(super.toString());
        sb.append("\n");
        com.ibm.etools.marshall.util.ConversionUtils.dumpBytes(sb, buffer_);
        return (sb.toString());
    }

    /**
     * @generated
     * @see javax.resource.cci.Record#getRecordName()
     */
    public String getRecordName() {
        return (this.getClass().getName());
    }

    /**
     * @generated
     */
    public byte[] getBytes() {
        return (buffer_);
    }

    /**
     * @generated
     * @see javax.resource.cci.Record#clone()
     */
    public Object clone() throws CloneNotSupportedException {
        return (super.clone());
    }

    /**
     * @generated
     * @see javax.resource.cci.Record#setRecordName(String)
     */
    public void setRecordName(String recordName) {
        return;
    }

    /**
     * @generated
     * @see javax.resource.cci.Record#equals()
     */
    public boolean equals(Object object) {
        return (super.equals(object));
    }
}

```

```

/**
 * @generated
 * @see javax.resource.cci.Streamable#read(InputStream)
 */
public void read(java.io.InputStream inputStream)
    throws java.io.IOException {
    byte[] input = new byte[inputStream.available()];
    inputStream.read(input);
    buffer_ = input;
}

/**
 * @generated
 */
public void setBytes(byte[] bytes) {
    buffer_ = bytes;
}
}

```

课程 1.4: 创建 Java 测试类以测试应用程序

课程 1.4 将指导您完成创建 Java 测试类以测试应用程序。

在开始之前，必须完成“课程 1.3: 创建消息缓冲区类”。在本课程中，您将完成下列任务：

- 创建 Java 测试类。
 - 使用下面提供的代码编辑该类。
 - 运行测试类来测试您的应用程序。
1. 首先，将创建 **Java 测试类**：展开 **MultiSegOutput** 项目，接着展开 **Java 资源**，然后选择 **sample.ims** 包。
 2. 右键单击并选择**新建**。选择  类选项以创建新的 Java 类。
 3. 在“Java 类名”字段中，输入 **TestMultiSeg**。注意，**TestMultiSeg.java** 类仅作为示例提供。根据 **IMS** 机器规范，可能需要更改事务代码。请咨询您的 **IMS** 管理员以获取事务代码。可在 **TestMultiSeg.java** 类中找到 **input.setIn_trcd("SKS6")** 语句并进行修改。
 4. 确保源文件夹包含 **MultiSegOutput/JavaSource**，包名包含 **sample.ims**。
 5. 单击**完成**。
 6. 在 Java 类编辑器中打开 **TestMultiSeg.java**。
 7. 使用下列代码替换编辑器中的所有代码：

```

/*****
 * Licensed Materials - Property of IBM
 *
 * com.ibm.j2c.cheatsheet.content
 *
 * Copyright IBM Corporation 2004. All Rights Reserved.
 *
 * Note to U.S. Government Users Restricted Rights: Use,
 * duplication or disclosure restricted by GSA ADP Schedule
 * Contract with IBM Corp.
 *****/
package sample.ims;

import com.ibm.etoools.marshall.util.MarshallIntegerUtils;
import sample.ims.data.*;

public class TestMultiSeg
{

```

```

public static void main (String[] args)
{
    byte[] segBytes = null;
    int srcPos = 0;
    int dstPos = 0;
    int totalLen = 0;
    int remainLen = 0;
    byte[] buff;
    short LL = 0;
    short ZZ = 0;

    try
    {
        // -----
        // Populate the IMS transaction input message with
        // data. Use the input message format handler method
        // getSize() to set the LL field of the input message.
        // -----
        InputMsg input = new InputMsg();
        input.setIn_ll((short) input.getSize());
        input.setIn_zz((short) 0);
        //-----
        // find out the transaction code from your IMS
        // administrator
        //-----
        input.setIn_trcd("SKS6 ");
        input.setIn_data1("M2 SI1");
        input.setIn_data2("M3 SI1");

        // -----
        // Run the IMS transaction. The multi-segment output
        // message is returned.
        // -----
        MSOImpl proxy = new MSOImpl();

        sample.ims.CCIBuffer output = proxy.runMultiSegOutput(input);

        // -----
        // Retrieve the multi-segment output message as a
        // byte array using the output message format
        // handler method getBytes().
        // -----
        System.out.println(
            "\nSize of output message is: " + output.getSize());
        segBytes = output.getBytes();

        srcPos = 0;
        dstPos = 0;
        totalLen = segBytes.length;
        remainLen = totalLen;

        // -----
        // Populate first segment object from buffer.
        // -----
        buff = null;
        // Get length of segment.
        LL =
            MarshallIntegerUtils.unmarshallTwoByteIntegerFromBuffer(
                segBytes,
                srcPos,
                true,
                MarshallIntegerUtils.SIGN_CODING_TWOS_COMPLEMENT);

        // Put segment in byte array.
        buff = new byte[LL];
        System.arraycopy(segBytes, srcPos, buff, dstPos, LL);
        remainLen -= LL;
    }
}

```



```

// Create and populate segment object from byte array.
OutputSeg1 S1 = new OutputSeg1();
S1.setBytes(buff);
System.out.println(
    "\nOutSeg1 LL is:  "
    + S1.getOut__ll()
    + "\nOutSeg1 ZZ is:  "
    + S1.getOut__zz()
    + "\nOutSeg1_DATA1 is: "
    + S1.getOut__data1());

// -----
// Populate second segment object from buffer.
// -----
srcPos += LL;
buff = null;
// Get length of segment.
LL =
    MarshallIntegerUtils.unmarshallTwoByteIntegerFromBuffer(
        segBytes,
        srcPos,
        true,
        MarshallIntegerUtils.SIGN_CODING_TWOS_COMPLEMENT);

// Put segment in byte array.
buff = new byte[LL];
System.arraycopy(segBytes, srcPos, buff, dstPos, LL);
remainLen -= LL;

// Create and populate segment object from byte array.

OutputSeg2 S2 = new OutputSeg2();
S2.setBytes(buff);
System.out.println(
    "\nOutSeg2 LL is:  "
    + S2.getOut__ll()
    + "\nOutSeg2 ZZ is:  "
    + S2.getOut__zz()
    + "\nOutSeg2_DATA1 is: "
    + S2.getOut__data1()
    + "\nOutSeg2_DATA2 is: "
    + S2.getOut__data2());
// -----
// Populate third segment object from buffer.
// -----
srcPos += LL;
buff = null;
// Get length of segment.
LL =
    MarshallIntegerUtils.unmarshallTwoByteIntegerFromBuffer(
        segBytes,
        srcPos,
        true,
        MarshallIntegerUtils.SIGN_CODING_TWOS_COMPLEMENT);

// Put segment in byte array.
buff = new byte[LL];
System.arraycopy(segBytes, srcPos, buff, dstPos, LL);
remainLen -= LL;

// Create and populate segment object from byte array.
OutputSeg3 S3 = new OutputSeg3();
S3.setBytes(buff);
System.out.println(
    "\nOutSeg3 LL is:  "
    + S3.getOut__ll()

```

```

        + "\nOutSeg3 ZZ is:  "
        + S3.getOut__zz()
        + "\nOutSeg3_DATA1 is: "
        + S3.getOut__data1()
        + "\nOutSeg3_DATA2 is: "
        + S3.getOut__data2()
        + "\nOutSeg3_DATA3 is: "
        + S3.getOut__data3());
    }
    catch (Exception e)
    {
        System.out.println("\nCaught exception is: " + e);
    }
}
}

```

8. 现在将测试应用程序：展开 **MultiSegOutput** 项目和 **sample.ims** 包。
9. 右键单击 **TestMultiSeg.java** 类并选择“运行”。选择“运行方式”>“Java 应用程序”。
10. 您应该会在控制台上看到以下输出：

```

Size of output message is:      99
OutSeg1 LL is:                  16
OutSeg1 ZZ is:                  768
OutSeg1_DATA1 is:               *****M1S01

OutSeg2 LL is:                  31
OutSeg2 ZZ is:                  768
OutSeg2_DATA1 is:               *****M1S02
OutSeg2_DATA2 is:               *****M2S02

OutSeg3 LL is:                  52
OutSeg3 ZZ is:                  768
OutSeg3_DATA1 is:               *****M1S03
OutSeg3_DATA2 is:               *****M2S03
OutSeg3_DATA3 is:               *****M3S03

```

11. 祝贺您！您已完成“多段输出”教程。

总结：为包含可变长度和多个段的 IMS 事务创建 J2C 应用程序

本教程将指导您完成一些详细步骤来生成一个 J2C 应用程序以处理可变长度和多段 IMS 事务输出消息。

已学习的课程

在本教程中，您将学习如何完成下列任务：

- 使用 J2C Java Bean 向导来创建运行 IMS 事务的 J2C Java bean。
- 创建消息缓冲区类 **CCIBuffer.java**，并使用 **doclet** 注释来编辑此类。
- 为 J2C Java bean 创建方法以运行 IMS 事务并为方法提供输入和输出数据类型。
- 为输出消息段创建 Java 数据绑定。
- 创建测试 Java 类 **TestMultiSeg.java** 以调用运行 IMS 事务的 J2C Java bean 方法，然后用 IMS 事务返回的数据填充缓冲区中的各个输出段。