CACI webMethods Developer Cook Book

Draft Version 1.0

Ray Moser

202-425-2360

09/30/2013

Date	Version	Update	Responsible
06/25/2013	0.01	Created outline and added content from John Kim	Ray Moser
06/28/2013	0.02	Added custom content starting with basic Flow coding; advanced Flow coding	Ray Moser
07/02/2013	0.03	Added debugging information	Ray Moser
07/03/2013	0.04	Added several sections and advanced Flow coding content	Ray Moser
07/03/2013	0.05	Updated Graphics; sent to Patrick Moore for draft review for work to date.	Ray Moser
08/02/2013	0.06	Cleaned up some of the text and added clarification where needed.	Ray Moser
08/26/2013	0.07	Continued adding new content	Ray Moser
08/28/2013	0.08	Continued adding new content	Ray Moser
08/29/2013	0.09	Continued adding new content	Ray Moser
09/05/2013	0.10	Continued adding new content; branch statements and started Loop steps.	Ray Moser
09/06/2013	0.11	Continued adding new content; continued Loop steps.	Ray Moser
09/09/2013	0.12	Finished Loop steps	Ray Moser
09/17/2013	0.13-0.16	Finished Repeat Step, Exit Step, Map Step	Ray Moser
09/18/2013	0.17	Started Prototyping Section 9.0	Ray Moser
09/23/2013	0.18	First major editing run; added appendixes information; Addressed Andy Kost's comments	Ray Moser
09/25/2013	0.19	Finished several major sections. Completed 2nd round of editing (more to come), added SVN.	Ray Moser
09/26/2013	0.20	Finished Jenkins Section, Added GIT Information	Ray Moser
09/30/2013	0.21	WmDB and JDBC Adapter	Ray Moser
09/30/2013	1.0	Final scrub and release of 1st version	Ray Moser

*Please contact Ray Moser, 202-425-2360 if you have any questions about this document.

TABLE OF CONTENTS

1.0 Document References 7
2.0 A Bit of History 8
3.0 Introduction 9
3.1 Document Purpose 9
3.2 Scope 9
3.3 Out of Scope 9
3.4 Audience 9
4.0 webMethods Integration Platform 10
4.1 Integration Server 10
4.2 Messaging (Broker – BR) 11
4.3 My WebMethods Server (MWS) 12
4.4 Designer/Developer 12
5.0 Coding/Naming Conventions 13
5.1 Packages 13
5.2 Folders 14
5.3 Services 15
5.4 Variables 15
6.0 Code Organization Convention 16
6.1 Development Folder Structure 16
6.2 webMethods Package Folder Structure 16
6.3 Service Structure 17
6.4 Coding Convention 17
6.4.1 General Guidelines 17
6.4.2 Flow Services 18
6.4.3 Java Services 18
7.0 Flow Coding Basics 20
7.1 Flow Invoke 21
7.2 Sequence 21
7.3 Map Step 22
7.4 Loop Step 22
7.5 Repeat/Retry Step 23

7.6 Exit Step 24
7.7 Branch Step 24
8.0 Advanced Flow Coding 26
8.1 Flow Invoke (Flow or Java) 26
8.1.1 Flow Properties – Package Navigator 26
8.1.2 Flow Properties – Tree View Tab 27
8.2 Sequence Step 32
8.2.1 Try/Catch Blocks 32
8.2.2 Sequences In Practice 33
8.3 Loop Step 34
8.3.1 One for One Loop Step Iterations 34
8.3.2 Narrow Down Data Sets with Loop Step Iterations 36
8.4 Branch Step 49
8.4.1 Evaluating Labels 49
8.4.2 Evaluating Pipeline Values 50
8.4.3 Using Regular Expressions (REGEX) to Evaluate Branches 52
8.5 Repeat Step 56
8.6 Exit Step 70
8.7 Map Step 70
8.8 Java Services: Concepts and Best Practices 72
8.8.1 Appropriate Use of Java Services 72
8.8.2 Java Services are Super-classed 72
8.8.3 Input/Output Signatures Should Always be Used 73
8.8.4 IData Utility 73
8.8.5 Internal Java Service Debugging 78
9.0 Recipes For Speed (Prototyping) 79
9.1 Use Pipeline Objects 79
9.2 Get Data Moving - Make XML Documents Quickly 81
9.2.1 How Humans View Data 81
9.2.2 How Integration Server Views Documents 83
9.2.3 Manually Create 83
9.2.4 XML Object Import 83
9.3 Flat File Documents 89

9.3.1 Create Documents Manually 89
9.3.2 Create Documents Automatically 89
9.4 Unit Testing for Results 95
9.4.1 Use Developer for Simple Step-Through Debug 96
9.4.2 Use the Administrative Console 96
9.4.3 Use the URL 98
9.4.4 Getting Data into the System – Use the Ports 99
10.0 Methods of Debuggery 100
10.1 Integration Server 100
10.1.1 Integration Server Logs 100
10.1.2 Server Log Settings 101
10.1.3 Server Error Log Settings 103
10.1.4 Custom Logs 103
10.2 Message Broker 105
11.0 Error Handling Framework 108
11.1 Error Handling Framework Overview 108
11.2 Necessary Steps to Implement IS Flow/Java Service Error Handling and Auditing 109
12.0 Testing 111
12.1 Unit Testing 111
12.2 System Testing 111
12.3 Integration Testing and User Acceptance Testing (UAT) 112
13.0 Java Database Connectivity (JDBC) 113
13.1 WmDB – webMethods Legacy Database Package 113
WmDB – Administrative GUI 114
WmDB – Flow Code 123
WmDB – Quickly Create or Drop DB Objects 126
13.2 JDBC Adapter 128
14.0 Using Source Control 129
14.1 Cooperative Development Practices 129
14.2 Subversion Change Management System 129
14.3 GIT Change Management System 129
14.3.1 GIT Best Practices 129
14.3.2 Basic GIT Workflow Example (from salesforce.com) 130

14.3.3 GIT Cheat Sheet Command Flow (Left to Right) 132
15.0 Deploying Your Code 133
15.1 Overview of webMethods Deployment 133
15.2 Standard Deployment 133
15.2.1 Integration Server 133
15.2.2 Broker Assets 134
15.2.3 BPM Models 134
15.2.4 Property Files 135
15.2.5 External Dependencies 136
16.0 Automated Deployment with Jenkins 137
16.1 Introduction to Jenkins Continuous Integration with webMethods 137
16.2 Installing Jenkins, webMethods Asset Build Environment (ABE), and Deployer 138
16.3 OS Level Configuration for Interoperability between Jenkins and webMethods 139
16.4 Configuring Jenkins 139
16.5 Configuring Jenkins Jobs 139
17.0 Luke's Linux Tips and Tricks 141
APPENDIX A: Built-In Service Review 142
APPENDIX B: Linux Quick Reference 148
APPENDIX C: Subversion Quick Reference 150
APPENDIX D: GIT Quick Reference for Common Commands 157

1.0 Document References

This Developer Guide is based on webMethods Suite version 8.2, Service Pack 2 (8.2.2). Documentation can be searched and retrieved from the Software AG web site or the entire library can be downloaded using Empower Credentials and the Download Manager.

Specific documentation will be referenced throughout this guide.

2.0 A Bit of History

The Software AG Integration Platform has been around since the mid-1990's. Phillip Merrick founded webMethods (original company) in 1996. Software AG (a German company) acquired webMethods in 2007 for $546 million.

Through the years, the company made a number of acquisitions using stock, cash or a combination of both. These acquisitions sought to bridge feature gaps in the platform features to meet customer expectations or but in many cases tried to propel the company ahead of the competition.

Today, the Software AG Integration platform represents a convergence of many of these products. The current solution offering provides users with an Eclipse-based IDE that consolidates multiple separate IDEs that were part of the product acquisition roadmap.

Users no longer need to use several IDE tools such as a developer, a modeler and others to develop, run or manage solutions. webMethods has converged all of the management and user interfaces under "My webMethods Server" (MWS) portal with the exception of the core Integration Server.

While the platform has undergone a number of significant changes that have stitched together the platform, the underlying mechanics remain largely the same. Under the hood, the Integration Server (IS) and Broker (BR) perform the lion's share of heavy lifting.

The Integration Server is considered the core component of the webMethods Integration Platform. It was built from the ground up to leverage XML messages. In fact, this design almost derailed the company during the early years. At the time, many companies used Electronic Document Interchange (EDI) and leveraged Value Added Networks (VANs) to transmit EDI documents to trading partners. webMethods released the WmEDI package early to address concerns that the platform was not ready for business. The EDI adapter represented one of many of add-on adapters that led the industry. With this adapter, customers managed EDI document-related transactions. A later addition, the WmEDIINT package provided AS2 functionality. The AS2 standard effectively removed the need for VANs and permitted EDI messages to be transported and managed using Internet protocols. webMethods has developed and provided over 80 adapters to meet customer integration needs for the major ERP, CRM and general B2B technologies of the era.

Today, the platform supports dozens of adapters. The most popular adapter is the JDBC adapter. SAG provides adapters for SAP, Mainframe Integration, FlatFile Integration and many others. SAG provides an Adapter Development Kit (ADK) that customers or IVRs may use to create custom adapters for proprietary systems or an API. The Momentum Adapter (MOMAPI) extends the ADK (a CGI offering) and is a solid example of a custom adapter.

3.0 INTRODUCTION

3.1 DOCUMENT PURPOSE

The purpose of this document is to provide best practices as they apply to integration application development using the webMethods platform. This book is laid out in a cook book fashion intertwined with introductory and advanced product information, and tips and tricks for various sections. For example, when the document details the different Flow code developer items (loop, repeat, etc.), the document provides multiple ways in which a developer can leverage the object. Where possible, the document details more than one method to accomplish a given task.

3.2 SCOPE

The initial document scope focuses on the development, debugging and testing of webMethods Integration Server Flow code. The cook book covers developer, Designer, Flow code, and Java code. It leverages the Broker.

3.3 OUT OF SCOPE

This document does not cover any application development standards, including Microsoft .NET, legacy applications, or any packaged applications other than webMethods Integration Platform.

Note: Process Design is out of scope for the initial version of this document but may be added later.

3.4 AUDIENCE

This document is intended for developers and development mangers concerned with the quality and uniformity of application code. The document assumes that the reader is acquainted with the webMethods platform and has some development experience. Advanced knowledge of the Software Development Lifecycle (SDLC) will help craft perspective.

4.0 webMethods Integration Platform

This section is a technical summary of the core components in use by your organization. This is not an overview of the entire product line but covers the major components in use.

4.1 Integration Server

Integration Server (IS) is a run-time server with a browser-based user interface whose function in application integrations is the execution of services. "IS" is a Java-based middleware application server. The server executes Java threads that execute Java programs whose primary focus is the data transfer by using Vector objects. SAG refers to the Vector object as the "pipeline". IS server is *XML* centric in that it was designed from the ground up to manipulate XML data at the smallest granular level. This is the major differentiator to this platform than from others such as IBM WebSphere (J2EE bean based) or Oracle WebLogic Server (J2EE-based).

Integration Server Supports	Examples
Transport standards	HTTP, HTTPS, FTP, FTPS, SMTP
Message formats	MIME, S/MIME
Data standards	XML and XML Schema, custom flat file formats with delimited fixed- or variable-length records
Protocols	SOAP, XML RPC, JMS
Specifications	Web Service Description Language (WSDL)
Integration patterns	REST, Web services, event-driven, request-reply

IS hosts packages that contain services and related files and comes with many packages that contain production-ready built-in services. For example, it includes services to convert date/time formats, file I/O, and client methods for ftp (put/get). The built-in services represent the type of processes or methods found in many development or scripting languages. Developers who leverage the built-in services save time that can be used later in the SDLC. Organizations create packages to contain the code, services or other middleware objects. Most items can be invoked (executed) from services residing in the IS, or by external calls.

IS performs the following duties:

- Runs services (mainly java-based)
- Handles authentication and authorization of clients
- Performs as a web server
- Maintains audit trail and logging
- Provides abstraction layer API for implementation of messaging
- Performance optimization (thread management, caching, etc.)

IS server is designed around the concepts governing service-oriented architecture:

- Fully supports the concept of reusable-services
- 4th Generation Language (4GL) Graphical UI development with drag and drop functionality that automatically generates Java code

- Fully extends Enterprise Application Integration through SOA standards (web services, flat files, EDI, and a myriad of adapters for SAP, Mainframe, Siebel, JD Edwards, etc.)
- Pipes and filters architecture pattern to divide large processing tasks into Sequences containing smaller steps
- Any services are interoperable as long as the inputs and outputs are matched

INTEGRATION SERVER DOCUMENTATION

Document	File Name
Administering Integration Server	8-2-SP2_Administering_Integration_Server.pdf
Developer Users Guide	8-2-SP2_Developer_Users_Guide.pdf
Integration Server Built-in Services Reference	8-2-SP2_Integration_Server_Built-In_Services_Reference.pdf
REST Developers Guide	8-2-SP1_REST_Developers_Guide.pdf
Service Development Help	8-2-SP3_Service_Development_Help.pdf
Web Applications Developers Guide	8-2-SP2_Web_Applications_Developers_Guide.pdf
Web Services Developers Guide	8-2-SP1_Web_Services_Developers_Guide.pdf

Note: This is a list of the most commonly used guides. Check the documentation folder in your installation or the Empower site for all of the documents available.

4.2 MESSAGING (BROKER – BR)

webMethods Broker represents the first acquisition (Active Software) made by the firm in the mid-1990's. Broker is considered core to the architecture and is one of the oldest components in the solutions suite. Originally, Broker was an EAI message bus. Later, it was revamped to meet the requirements to provide JMS messaging. It is a high-speed message router that uses a browser-based interface for administration and configuration. An original command-line-interface (CLI) is available. Broker provides for point-to-point and publish/subscribe messaging through a number of different standard message patterns:

- Publish and Subscribe – Publishers publish documents to the Broker and subscribers (consumers) receive them. (asynchronous)
- Publish and Wait – Publishers publish documents and wait for a reply (synchronous)
- Publish and Deliver – Publishers publish documents to a specific destination or client ID (synchronous – real time).

The Broker can execute messages synchronously or asynchronously using built-in methods. To receive a message, a client must connect to the system. Integration Server uses trigger objects to facilitate client connections. webMethods provides APIs to create clients using Java and C#.

BROKER DOCUMENTATION

Document	File Name
Administering Broker	8-2-SP3_Administering_Broker.pdf
Broker Messaging Programmers Guide	8-2-SP2_Broker_Messaging_Programmers_Guide.pdf
Publish/Subscribe Developer Guide	8-2-SP2_Publish_Subscribe_Developers_Guide.pdf

Note: This is a list of the most commonly used guides. Check the documentation folder in your installation or the Empower site for all of the documents available.

4.3 My WebMethods Server (MWS)

My webMethods Server (MWS) serves as the platform administration and configuration interface. It also provides a portal to deploy custom applications developed using the Composite Application Framework (CAF – outside the scope of this document). MWS brings a lot of capabilities to the Enterprise. This document focuses on the administration and configuration aspects that relate to Broker, Process Models and Integration. MWS provides wiki, forums and other enabling tools that are beyond the scope of this document.

It's easier to note what MWS does not provide: Integration Server and Deployer administrative interfaces and interfaces for adapter packages hosted in Integration Server. These older interfaces have not been migrated to MWS. Users will need to read the instructions for each adapter, Integration Server or Deployer to determine the URL needed to open up the respective interface.

MWS brings a totally new view for Broker users. Now, users are able to view client groups, clients and documents. Prior to MWS, the only way to view this information was through custom Java services that leveraged the Broker Java/C# API.

Users and administrators previously used another tool to view process models. This feature was consolidated into MWS. Now process model activity can be viewed in real time and replayed (if configured).

MWS Documentation

Document	File Name
Administering My WebMethods Server	8-2-SP1_Administering_My_webMethods_Server.pdf
Diagnosing My WebMethods Server	8-2-SP1_Diagnosing_My_webMethods_Server.pdf
Diagnosing My WebMethods Server Supplement	8-2-SP1_Diagnosing_My_webMethods_Server_Supplement.pdf
Working with My WebMethods Server	8-2-SP1_Working_with_My_webMethods.pdf

Note: This is a list of the most commonly used guides. Check the documentation folder in your installation or the Empower site for all of the documents available.

4.4 Designer/Developer

webMethods Developer was an early 4th Generation Language (4GL) IDE tool that provided a graphical environment for developers to code complex java. By using drag-and-drop component building and mouse-drags to map data points from source to target, Developer brought true rapid development concepts to users.

With the exception of 1-2 minor tasks, Developer has been deprecated in platform version 8.2.2. It is still worthy of mention because it still serves a very useful purpose in debugging efforts and historically, there are a lot of product ties to this application. Developer was a Java application written in C++ and optimized for Windows.

Designer is an Eclipsed-based replacement for Developer. Another legacy IDE, Modeler (used to develop process models) was aggregated into Designer as a plug-in.

Designer should be used for all software development tied to the webMethods platform. This includes IS Services, IS Java Services, BPM Process Modeling, Task Development (workflow), Composite Application Framework (CAF), Web Services, and Centrasite.

Tips and Tricks – Setting Optimal Designer Properties

Navigate the menu by choosing Windows >> Preferences >> Software AG >> Service Development

- Choose >> Flow Service Editor – Add/Remove "favorite" Services that Designer will list in the Designer palate.
- Choose >> Java or "C" Service Editors – Choose which Java signature needed. Java services that are imported from older packages (pre-8.x) use a different signature that may introduce compilation errors. This has to do with extra spaces in the pre-8.x code that the new Designer interface does not allow. The fix is to open it up in Developer and remove the extra spaces.
- Choose >> Package Navigator – Uncheck the box labeled "Hide generated Flow services". This will permit you to view Packages automatically generated by Process Models.
- Choose >> Schema Editor – to add or remove Schema expansion level in Designer
- Designer installs with a 1.6.x version of Java (as of July 2013)
- Try to leverage the same JDK for developing in Designer as the Integration Server
- **If Designer windows or tabs get out of control, Choose >> Window >> Reset Perspective**. This returns the Designer GUI to the default window setting.

DESIGNER/DEVELOPER DOCUMENTATION

Document	File Name
Developer Users Guide	8-2-SP2_Developer_Users_Guide.pdf
Integration Server Built-in Services Reference	8-2-SP2_Integration_Server_Built-In_Services_Reference.pdf
REST Developers Guide	8-2-SP1_REST_Developers_Guide.pdf
Service Development Help	8-2-SP3_Service_Development_Help.pdf
Web Applications Developers Guide	8-2-SP2_Web_Applications_Developers_Guide.pdf
Web Services Developers Guide	8-2-SP1_Web_Services_Developers_Guide.pdf

Note: This is a list of the most commonly used guides. Check the documentation folder in your installation or the Empower site for all of the documents available.

5.0 CODING/NAMING CONVENTIONS

This section describes naming convention guideline for packages, folders, services, and variables.

5.1 PACKAGES

- Capitalize the first letter of the package name. If a name consists of many words, capitalize the first letter of each word (Upper CamelCase aka Pascal Case). For example, a package that contains purchase order logic could be named: PurchaseOrder. Package names should always be capitalized. If a name is composed of many words, capitalize the first letter of

each word. An underscore could be used to separate key words as well. (Example: FEDEX_OrderResponse).

- Do not use non-alphanumeric characters. In other words, do not use special characters or symbols. It is acceptable to use an underscore "_" to break apart words such as PO_X12_850 (EDI purchase order), PO_IN_850 (inbound), PO_OUT_850, X12_855, etc. But this is an architecture decision.
- Give the package a name that reflects its purpose of the function or the services it contains. Avoid using generic names such as "Development."
- Give the package a name that is appropriate for both the package and the top-level folder. The top-level folder that is visible in Designer starts the namespace. The actual package name is ignored by webMethods.
- Never prefix your package name with "Wm". This is a key naming convention used by the Integration Server to denote vendor-distributed packages.
- Avoid common package names such as "Utilities", "Common", or "Global".
- When creating versions and releases for a package, do not include the version or release number as part of the package name.
- Do not include the work "Package" or "Pkg" at the end of the package name
- If one or more of the words in the package name is an acronym, it is generally a good idea to capitalize only the first letter (Example: SapBwRfc rather SAPBWRFC).
- Avoiding creating package names with random capitalization: For Example, mYPkgTest.
- Avoid using articles (e.g., "a", "an", "the") in the package name. For example, instead of "TestTheService", use "TestService."
- The package names should not contain any characters that may cause problems for any operating system. Avoid using any non-alphanumeric characters.
- Document the package by creating HTML documents that state the purpose of the package. Place these documents under the /pub directory in the package and the documents will deploy with the package. Make sure an "index.html" page is created as the base or home page.

5.2 FOLDERS

- webMethods Integration Server is java-based and utilizes java-compliant name space conventions. The Package names are not visible to the application space. It is the first level of folders that are viewed by the application space (corresponds to an interface.) Key words should not be used to name top level folders: File, Enumeration. If required, use the lowercase folder name: file, enumeration.
- Folder names should be capitalized: "Utility.Admin." If a name is composed of many words, capitalize the first letter of each word. An underscore could be used to separate key words as well. (Example: CompanyA_OrderEntry) If there will be a large amount of folders, than an underscore will produce a human-readable format. This corresponds to name space

naming conventions. If a name consists of many words, capitalize the first letter of each word (Upper CamelCase aka Pascal Case.)

- Create only one top-level folder in each package and ensure that its name matches the package name. This prevents namespace conflicts (i.e. "<package1>/pub/service" and "<package2>/pub/service".)
- Integration Server creates Java classes when Java Services are created. All of the Java services at a given level in the folder hierarchy are super-classed together. Therefore, do not capitalize a folder name when it conflicts with an existing Java class name (for example, File, Enumeration.) In these cases, use a lowercase folder name (for example: file, enumeration).

5.3 SERVICES

- Service names should start with a verb, in lowercase, and any subsequent objects in a service name should be capitalized. Capitalize any subsequent objects in a service name (Lower CamelCase) (i.e., holdObligation, processCommitment). This is consistent with Java naming conventions for methods.
- Try to stay consistent with the verb usage (i.e., map vs. transform vs. set, get vs. select, validate vs. verify).
- Provide a name that is as descriptive as possible.
- Non-Flow and Java services may not start with a verb, but should still use Lower CamelCase.

5.4 VARIABLES

- Follow the generally recognized Java variable naming convention, which dictates that the first letter of each variable is lowercase and each subsequent word is capitalized (i.e., documentNumber). Although Integration Server will allow variables to be named using non-Java standards such as the use of special characters, this practice is discouraged.
- Define global variables in uppercase to easily differentiate them from other variables (for example, DB_ALIAS).
- Referenced IData object (webMethods document) variables should use the name of the original IData object name, unless a different name is needed. This allows for easier implicit mappings so explicit mappings are not required.
- Variable data extracted from RDBMS may be declared in upper case: DTD_RECEIVED (date received), DT_RXD (date received). This serves to quickly distinguish the source of the data for developers and support personnel.

6.0 Code Organization Convention

This section describes code organization conventions. This folder structure resembles what a developer can see in the Designer, or what an administrator may see when browsing services in the administrator view. This section is compliant with current project practices.

6.1 Development Folder Structure

Standard folder structure underneath the highest folder level of the Integration Server package. USERS create these folders. These are not the same as the folders that the underlying system creates.

- <InterfaceName>
  - docs
  - interface
    - flatfiles
    - triggers
    - webservices
  - model
    - impl
  - services
    - jdbc
    - momapi
  - utils

6.2 webMethods Package Folder Structure

Understanding how packages are structured by the Integration Server will help with debugging code and preparing packages for deployment. The folder names viewed represent the actual folders created by the system for a given package. It is not directly representative of what can be viewed in the Designer. The ./ns folder contains the paths that are typically displayed in the Designer and contains .xml run-time descriptor files that build the 4GL view that Designer renders.

The following table depicts the file folder structure for a given package and is consistent with current project practices. It further details the types of files expected in each folder.

- <InterfaceName> - top level folder name
  - code – Java source and class files
  - config – property files

- doc – readme files, documentation
- lib – jar files
- ns – Flow code files
- resources
  - bpm – contains .process files, build.xml file
  - Broker – contains Broker xml files
  - test
    - data – unit test input/output xml data
    - setup – unit test suite files
    - script – unit test task scripts

6.3 Service Structure

- All services that are generic and can be used across various interfaces should be placed in a common namespace location (i.e., common.utils.math:addDollarAmounts).
- Do not divide code into packages/namespaces based solely on developer. Always structure code by function. One case that supports splitting or dividing code between namespaces is when developing a fix or patch that is deployed as a separate package that contains separate code but must exist in an existing namespace. The Integration Server will not allow duplicate services in the same namespace.

6.4 Coding Convention

This section describes coding convention guidelines for writing Flow and Java services.

6.4.1 General Guidelines

- Use the built-in services when possible. They have been tested and proven.
- For services not provided by the vendor, write low level utility and "often used" services in Java if possible. Java services have slightly better performance than Flow services.
- Flow services should be used to bring together various types of services and in places where code can be easily visualized.
- Code utility services to perform only one function (if you cannot come up with a good name for the service, you may want to break it up into two services.)
- Non-string or IData objects can be set in the inputs/outputs of a service by using a generic Object variable. You can specify a primitive type wrapper object in the property, otherwise it will attempt to type cast it when used.
- Be cognizant that copying and pasting variables also copies its metadata (i.e., constraints, data type, etc.)
- Be cognizant that copying and pasting services also copies its metadata (i.e., cache, audit, logging, etc.)

- Always drop variables that are no longer used from the pipeline to reduce clutter and memory footprint. This can be done on a step-by-step basis or in a map step at the end of a section or an entire Flow. The recommendation is to establish a pattern and stick to the pattern to make debugging easier. A common problem occurs when needed variables are dropped. This usually throws a null pointer exception.
- Parent services can call child services in three methods:
    - Flow Invoke: create a shallow clone of whole pipeline object and merges that pipeline object with the output pipeline object. This uses the **HIGHEST MEMORY OVERHEAD.**
    - Java – sends the whole pipeline object and returns the same pipeline object – **LOWEST MEMORY OVERHEAD**.
    - Transformer – creates a new pipeline object with only input variables and merges that pipeline object with the output pipeline object – **MINIMAL MEMORY OVERHEAD**.
    - The architecture should take into consideration planned and future system growth, projected through-put and expected or agreed upon performance. Developing a lean architecture pattern is high desirable but always determine if the proposed solution will address the underlying business issue, solve the problem, or meet the requirements. At some point during the development, if the adopted approach will not address the requirements, then a meeting with stakeholders should take place and other potential avenues of approach discussed.

6.4.2 Flow Services

- Comment each step in the Flow service. This aids the system to document code for future developers and operations and maintenance staff. They are especially helpful for collapsible steps like Sequences, Loops, etc.
- When using Map Flow objects, always set a comment to describe what is occurring inside the map since they are not self-documenting.
- Placing a map object at the beginning of a Flow to set pipeline variables is a best practice.
- Dropping objects at the end of a Flow service as opposed to dropping in the middle of a service is a common and considered a best practice. Developers may need to deviate from this to remove large document objects from the pipeline to enhance or optimize the pipeline for high volume transactions. It is best to adopt a standard approach across all of the organization code to make debugging and maintenance more efficient.

6.4.3 Java Services

- The use of Java services initiates the creation of a folder to contain the underlying Java source and class files. The Java service becomes a method to the superclass. The developer should set the input and output signatures to the service. To gain access to these variables, use the IData object and IData cursor to get or put the data from/to the pipeline.
- In the Java class, other methods can be created and used without creating a Java service. These methods do not appear as a Java service in the Integration Server.

- Any Java methods that are intended to be used across interfaces as a service should be created as a Java service so that it is easily accessible from Flow services. This also supports the paradigm of reusability.
- The IDataCursor object needs to be created to pull objects out of the IData pipeline. The IDataCursor object should always be destroyed at the end of the Java service.

Tips and Tricks – Java Services

- Java services are super-classed. All Java services for a given folder level are super-classed.
- Java services that are imported from older packages (pre-8.x) use a different signature that may introduce compilation errors. This has to do with extra spaces in the pre-8.x code that the new Designer interface does not allow. The fix is to open it up in Developer and remove the extra spaces.
- Designer installs with a 1.6.x version of Java (as of July 2013)
- Try to leverage the same JDK for developing in Designer as the Integration Server

7.0 Flow Coding Basics

The Integration Server has not changed the basic object structure since bringing Integration Server to market. Even though the "Developer" IDE is deprecated, the Eclipse Designer (that replaced Developer) has continued the tradition. This makes transitioning to the new IDE easy for most and even easier for those with some familiarity with Eclipse.

Developers use a drag and drop Java GUI framework to write or code Flow service language. It is implemented using an older, but very stable version of Eclipse. Many developers create empty code structure (with try/catch blocks and pointers to internal error handling) and then cut and paste the empty code structure into their new Flow services as a time saver.

Flow code contains several Flow steps. A Flow Service is considered a basic unit of work and is targeted for unit testing. The webMethods Flow language provides a set of control steps that allow you to direct the execution of a Flow service at run time. The control steps allow you to:

- Conditionally execute a specified sequence based on a field value
- Retry a specified sequence until it succeeds
- Repeat a specified sequence (loop) for every element in a given array field.

The following sections contain the basic Flow step objects:

7.1 FLOW INVOKE

The Flow invoke is the most commonly used object and executes a specified Flow service.

Image	Name	Description
➡	Flow Invoke	Flow symbol represents a child-call. Contains the path statement for name: namespace.name:**FlowName** For example, the built-in "**concat**" function that concatenates 2 strings together is called by invoking pub.string:**concat**. The namespace name is separated by periods but the final service is separated by a colon ":".
Property Setting		**Description**
Comments		Add comments in this field to describe the map step; these help to provide documentation for the overall Flow service;
Scope		Specifies the name of a document (IData) in the pipeline that you wish to restrict to this service. To submit the entire pipeline, leave blank.
Timeout		Specifies how long, in seconds, this step should run. If the time elapses, the server will raise an exception after the step completes.
Label		Specifies a label for this step, for use with BRANCH or EXIT step.
Typical Uses		Must be used for every Flow or Java service invocation. Java services show up with this symbol within Flow code.

7.2 SEQUENCE

Image	Name	Description
	Sequence	Used to group a series of steps together.
Property Setting		**Description**
Comments		Add comments in this field to describe the map step; these help to provide documentation for the overall Flow service;
Scope		Specifies the name of a document (IData) in the pipeline that you wish to restrict to this service. To submit the entire pipeline, leave blank.
Timeout		Specifies how long, in seconds, this step should run. If the time elapses, the server will raise an exception after the step completes.
Label		Specifies a label for this step, for use with BRANCH or EXIT step and does not contribute to the SEQUENCE step.
Exit on		Specifies when to exit the SEQUENCE step. Select FAILURE to exit when a child step throws a server error, Choose SUCCESS to exit child steps when all steps execute successfully without error. Choose DONE to exit the SEQUENCE after all steps have finished (even if there are issues.)
Typical Uses		This is used to formulate a framework to facilitate error handling. It recreates the try/catch paradigm. See the advanced Flow code SEQUENCE section. In large or

	complex Flow services, Sequences can be used to group Flow services.

7.3 MAP STEP

Image	Name	Description
	Map	Map steps allow developers to perform editing operations on pipeline objects. This can include mapping data values from source (left side) to the target (right side), adding variables to the pipeline, and dropping variables from the pipeline.
Property Setting		**Description**
Comments		Add comments in this field to describe the map step; these help to provide documentation for the overall Flow service;
Scope		Specifies the name of a document (IData) in the pipeline that you wish to restrict to this service. To submit the entire pipeline, leave blank.
Timeout		Specifies how long, in seconds, this step should run. If the time elapses, the server will raise an exception after the step completes.
Label		Specifies a label for this step, for use with BRANCH or EXIT step. The map object is typically one of several child objects of the BRANCH or EXIT step.
Typical Uses		Used to add, drop or map data in the pipeline. Transformers are available for use in this Flow step. **SEE ADVANCED CODING for more on transformers.**

7.4 LOOP STEP

Image	Name	Description
	Loop	Executes a set of Flow steps once for each element in a specified array (looping).
Property Setting		**Description**
Comments		Add comments in this field to describe the map step; these help to provide documentation for the overall Flow service;
Scope		Specifies the name of a document (IData) in the pipeline that you wish to restrict to this service. To submit the entire pipeline, leave blank. **This is important for LOOP Steps because it reduces the amount of memory overhead but cannot always be used if other pipeline variables are needed during the Loop operation.**
Timeout		Specifies how long, in seconds, this step should run. If the time elapses, the server will raise an exception after the step completes.
Label		Specifies a label for this step, for use with BRANCH or EXIT step.
Input Array		Specifies the input array over which to loop. You must specify a field that is an array data type such as String list,

	String table, document list, or Object list.
Output Array	Optional. Specifies the name of the field containing output data for each iteration of the loop. It collects the output from the iterations into an array field with the same name.
Typical Uses	If an input and output array are named, then it is assumed that the code seeks to map 1-for-1. The output array will contain the same number of items as the input array. In the instance where only an input array is specified, the code typically contains code to filter the output. Since the output contains less fields than the original (filtered), it is inappropriate to map into an output array. In this case, the developer will need to map the data into a single instance of a group of field data (document) and then use the built-in services to append to a document list. See the advanced Loop section for more information.

7.5 REPEAT/RETRY STEP

Image	Name	Description
	Repeat/Retry	Re-executes a set of Flow sets up to a specified number of times based on the successful or unsuccessful completion of the set.
Property Setting		**Description**
Comments		Add comments in this field to describe the map step; these help to provide documentation for the overall Flow service;
Scope		Specifies the name of a document (IData) in the pipeline that you wish to restrict to this service. To submit the entire pipeline, leave blank.
Timeout		Specifies how long, in seconds, this step should run. If the time elapses, the server will raise an exception after the step completes.
Label		Specifies a label for this step, for use with BRANCH or EXIT step.
Count		Specifies the maximum number of times the server re-executes the child steps subordinate to the REPEAT step. It does not count the first iteration. So, to execute 50 times total, this value should be 49.
Repeat interval		Specifies how long the server waits before re-executing the child steps. The default unit is in seconds. If left blank, no interval - instant.
Repeat on		Choices are SUCCESS or FAILURE. Specifies when the server re-executes the REPEAT child steps. Select SUCCESS to re-execute when ALL of the child steps execute. Select FAILURE to re-execute when any ONE of the child steps fail.
Typical Uses		This represents a "Do While…" condition. This has been used to facilitate reconnections (failure) to external systems (adapter connections.) It is also used in large file parsing activities. See advanced Flow code for more information.

7.6 EXIT STEP

Image	Name	Description
	Exit Step	Allows the code to exit based on specific pipeline information that is set at design time.
Property Setting		**Description**
Comments		Add comments in this field to describe the map step; these help to provide documentation for the overall Flow service;
Label		Specifies a label for this step, for use with BRANCH or EXIT step.
Exit from [$Loop | $parent | $Flow | Label]		Specifies the Flow step or service that you wish to exit from. If **$Loop** is selected, it will exit from the Loop or repeat object when the condition is met. If it is a child of a Repeat step and a particular condition is met, then the EXIT step will break the condition of the REPEAT step. If **$parent** is selected, it will exit from the parent Flow service. If this is exit (on FAILURE), then it will bubble up an error to the parent Flow to handle. If it is set to exit from **$Flow**, it will exit from the Flow step that contains the exit step. If **Label** is set, then it will exit from the nearest object that contains the named label. If label is set and there is no label, it will throw an error.
Signal [SUCCESS | FAILURE]		Specifies whether the exit is considered a success or failure. Select FAILURE to exit and throw an exception. Select SUCCESS to exit without an exception.
Failure message		Specifies the exception message displayed when the EXIT step signals a failure.
Typical Uses		It is used normally to exit from a Loop or repeat object once a condition has been met, but it can exit from Flows, parent Flows, or an object with a named label.

7.7 BRANCH STEP

Image	Name	Description
	Branch Step	Evaluates a specific Flow step based on the value of a specified variable in the pipeline. May use regular expressions
Property Setting		**Description**
Comments		Add comments in this field to describe the map step; these help to provide documentation for the overall Flow service;
Scope		Specifies the name of a document (IData) in the pipeline that you wish to restrict to this service. To submit the entire pipeline, leave blank.
Timeout		Specifies how long, in seconds, this step should run. If the time elapses, the server will raise an exception after the step completes.
Label		Specifies a label for this step, for use with BRANCH or EXIT step.
Switch		Specifies the String field that the BRANCH step uses to

	determine which child field and what the field value is in order to execute the step. NOTE: THIS IS ONLY USED IF the field EVALUATE Labels property is set to FALSE.
Evaluate Labels [true | false]	**If true**, then the labels applied to the subordinate Flow objects are evaluated based on a field value. (Switch field is blank if Evaluate Labels is true). **If true**, can use Regular Expressions to define value. **If false**, then the Flow control is switched based on the value of the Switch field. .
Typical Uses	Used to evaluate which subordinates Flow step to invoke based on configuration. Only allows for 1 specific logical step to execute. See Advanced Flow Code for specific examples.

8.0 Advanced Flow Coding

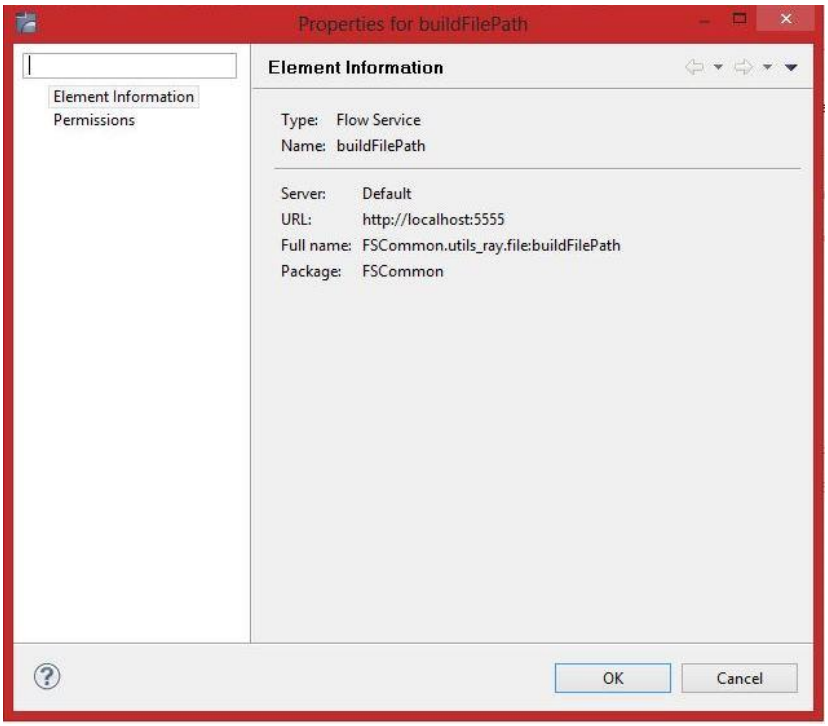
Reorder to match previous section.

8.1 Flow Invoke (Flow or Java)

The Flow invoke is the most commonly used object and executes a specified Flow service. Each Flow or Java service has specific property settings. These properties are different than those covered in the previous section. To make things more confusing, there are several sets of properties, all called properties.

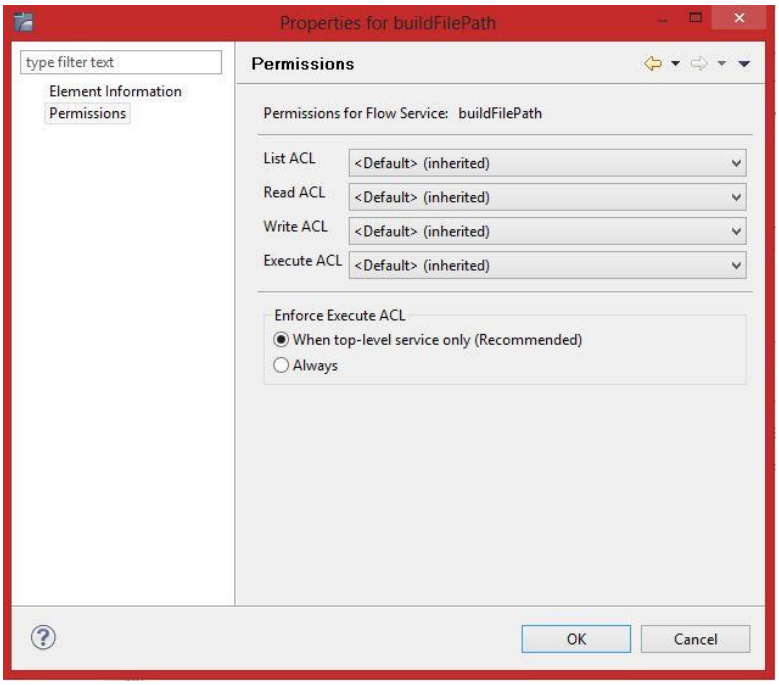
8.1.1 Flow Properties – Package Navigator

Open the Package Navigator Tab (normally on the left side of the Designer) and right mouse click on a service to obtain the properties or hit ALT + ENTER. This action will produce a popup window with two choices to select, Element Information (containing basic non-editable information about the service), and Permissions (containing ACL information). Nothing can be modified without locking the service first. Also note, that if the property tab in Designer is open, it will be populated with the same information. The following graphic depicts the expected popup window with Element Information:



No information may be edited in this window.

The following graphic depicts the properties window with “Permissions”:



Each service may be configured individually to limit List, Read, Write and Execute permissions. The Default setting is "<Default> (inherited)". The "Enforce Execute ACL" choices at the bottom designate whether or not to enforce the Execute ACL. It is recommended that if the service is a parent or top level service, that the Execute ACL is configured to a specific user and enforced. Remember, to modify the settings, the service must be locked.

8.1.2 Flow Properties – Tree View Tab

To obtain access to the properties for a given Flow service, open the Flow service in Designer and make sure that the properties tab is open. Click anywhere on an empty spot on the Tree View Tab and the property tab (normally on the right will open).

The properties that display from this activity are the lowest level properties available for a Flow or Java service that is configurable through the Designer.

The following graphic depicts the various properties or settings that are available for each Flow service. To access the properties for a given Flow service, click on the blank space in a Flow service's TREE view. If the property tab is open, the properties for the Flow service will display.

Outline | Properties

Property	Value
General	
Permissions	
Comments	
Reuse	Private
Source URI	
Run time	
Stateless	False
Cache results	False
Cache expire	15
Reset cache	
Prefetch	False
Prefetch activation	1
Execution locale	[$null] No Locale Policy
HTTP URL alias	
Pipeline debug	None
Transient error handlin	
Max retry attempts	0
Retry interval	0
Audit	
Enable auditing	Never
Log on	Error only
Include pipeline	Never
Universal name	
Namespace name	
Local name	
Output template	
Name	Default_testFlow
Template source	

The following table breaks down the properties and provides a description of how the property is used.

FLOW and JAVA SERVICE PROPERTIES

Property Setting	Description	Values
General		
Permissions	Set the ACL permissions for List, Read, Write and Execute	For each type of ACL, the drop box displays a list of users. Select the user with the correct permissions.
Comments	Allows user to store comments about the service.	Open text. Recommend limiting to 100 characters or so since it will not be readable in the Designer.
Reuse	Indicates if the asset is intended for internal reuse (Private) or external reuse (Public). If Public is selected, the asset may be shared through the CentraSite registry (UDDI) with a BPM or CAF project.	Private | Public
Source URI	Location of the source for this asset. It will normally be blank. If the Flow service was created using an XML document, an XML schema or a WSDL file, then the URI pointing to the document will be listed.	See http://www.w3.org/Addressing/
Run time		
Stateless	Default setting is false. If set to true, the service runs stateless which results in faster execution under load. Under the covers, it specifies whether the server removes the client session and its session object immediately after service execution is completed.	True | False
Cache results	By caching results, it will reduce memory usage. This property should only be set true for static inquiries. Dynamic queries will never be cached. The server will cache results for those queries containing the same exact inputs.	True | False
Cache expire	If results are cached, indicates when the cache expires and the query should be run again. This occurs only on-demand and is not automatic (see pre-fetch below). It is an integer, in minutes that will expire the cache.	True | False
Reset cache	This is a Designer-only function.	Manually resets the cache for that

	Press the reset button and it will reset the server cache. To reset the cache without using the Designer, open the "browse services in a package" functionality in the administrative console. Click on the link to reset cache.	service.
Prefetch	Default setting is false. When set to "True", specifies that the server automatically refreshes a cached result when it expires by re-executing the service using the same inputs.	True | False
Prefetch activation	Specifies the minimum number of times a service must be accessed in order for results to be prefetched.	The default setting is "1".
Execution locale	Default setting is: [$null] No Locale Policy. Choose the default locale needed for this service. This is related to I18N.	[$null] No Locale Policy
HTTP URL alias	The HTTP URL path alias for the default HTTP invoke URL path. The path portion of the URL consists of the invoke directive and the fully qualified service name.	Blank
Pipeline debug	The default setting is None. This setting permits the developer, operations and maintenance staff or other interested individuals to save or restore pipeline values. The system only saves the resulting pipeline. The settings can be overwritten at the global server level by the settings indicated in: watt.server.pipeline.processor	• None – Does not save • Save – Saves pipeline • Restore (override) – Over writes previous restore • Restore (Merge) – Merges previous restores.
Transient error handling		
Max retry attempts	Specifies the number of times the Integration Server will attempt to retry the service when the service fails due to an intransient error that results in an ISRuntimeException. Transient errors arise from conditions that may be resolved quickly such as the unavailability of resources due to network issues or the failure to connect to a database.	Default is "0".
Retry interval	In milliseconds, it specifies when the	Default is "0".

	server will re-execute the service.	
Audit		
Enable auditing *Note: Can have an adverse effect on performance.*	The default setting is Never. If Top Level Service is selected, the server collects audit data when it is triggered by a task, client or other method. If a service makes the call, then the service is not acting as a top level service and audit will not occur.	• Never • When Top Level Service Only • Always
Log on	This specifies when logging occurs. Typically, only Error information is logged. There may be business reasons to capture more information. Be aware that the more information that is targeted for audit, the more memory resources are required. A substantial number of activities execute when an error occurs so performance degrades substantially and could impact overall operations. If you are only concerned with services that fail, then select Error.	• Error - Only if service throws an error • Error and Success - If a service runs successfully, or fails • Error, Success and Start - Logs every possible condition
Include pipeline	"Never" indicates that the pipeline will never be captured and included in the service log. On Errors Only indicates that the pipeline will be captured in the event of an error. This is a helpful setting when there are errors that are difficult to debug. The "Always" setting captures a pipeline image for each execution. This selection can be resource intensive, however, if you plan to retry a service using the webMethods Monitor, this must be selected.	• Never - Default • On Errors Only - A good way to capture a COMPLETE view of the data in the pipeline. • Always - Not recommended since it can cause an adverse effect on performance.
Universal Name		
Namespace name	An external identifier that external protocols, such as SOAP, use to reference an Integration Server Service. This must be an absolute URI and not relative. Relative URIs are not supported.	Freehand input of the Namespace name.
Local name	This unique identifies a service within the namespace name.	Freehand input of the local name.
Output template		
Name	Assigns a name to the template.	Dynamic Server Pages (DPS) use

		output templates to render HTML content.
Template source	Opens up a text box to allow for entry or modification to the template.	Location of template on file system. Typically in the package structure under pub.

Tips and Tricks – Flow Service Properties

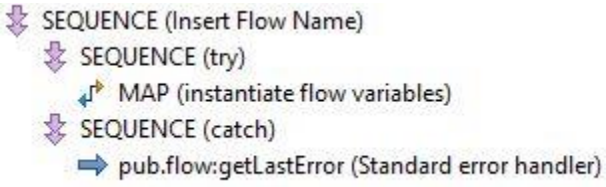
- If no properties are set for ACL, they will be inherited.
- Running stateless services are more efficient especially when a special user executes (a machine user versus a logon user). Audit captures user information.
- Most services should run stateless. This insures that variables will not be cached by the server.
- Caching can potentially introduce inaccurate data. The determining factor is if the results will always return the same information. Caching can significantly speed up the return of reference or static data. It should never be used when data is dynamic in nature.
- The use of service level caching should be compared to using the Terracotta in-memory big data caching. Service level caching uses the Integration Server's JVM heap and is limited by that setting Terracotta runs under another JVM PID that is substantially larger.
- Make them testable – Flow services should be callable by any other service. This means that inputs and outputs must be specified. This also permits the use of the Flow service in a transformer.
- Write comments whenever possible, but use a defined and acceptable pattern.

8.2 Sequence Step

Use of Sequences – Sequences should be used to build a set of steps that should be executed as a group. Each Flow step will be executed in the order in which they appear in the sequence. The exit-on property for Sequences is the most important.

8.2.1 Try/Catch Blocks

By grouping the Sequences as indicated in the graphic below, a try/catch block will be produced:



The parent sequence, labeled (Insert Flow Name) has an exit-on property of SUCCESS. The first child sequence labeled (try) has an exit-on property of FAILURE. The second sequence labeled (catch) has an exit on property of DONE. When the Flow containing this parent/child sequence structure begins, the first child sequence (try) starts. If the sequence or grouping of code runs successfully, then the sequence is complete and the second (catch) sequence never starts. If, however, an error is thrown during the time when the thread is running in the first child sequence (try), then the second sequence begins. This is because the exit-on property for the first child sequence (try) is set to FAILURE. The second child sequence (catch) has an exit-on property of

DONE, so all items will complete before the sequence is finished. Once these tasks are complete, the parent exit-on property of SUCCESS has been met and the parent sequence is complete. Sequence groupings are an integral part of error trapping and the adherence to using these structures is very important.

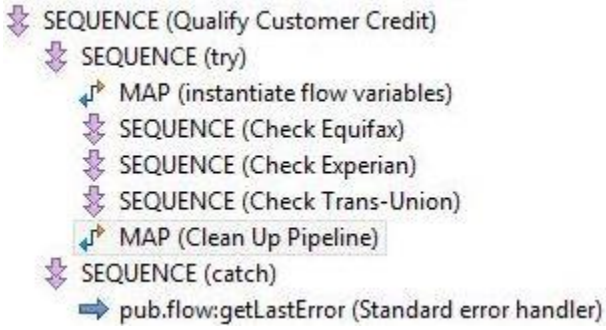
All top-level Flows should use error standard handling procedures. Depending on the type of error (expected, unexpected), appropriate notification processes will trigger. Error handling can be obtained by grouping the Sequences as above or by other means. This may not apply to Flow service process steps since the process model defines/possesses an error handler.

8.2.2 Sequences In Practice

Some Flow services contain a large number of steps by design. In an effort to make the Flow code readable and easier to manage, developers can use Sequences to group Flow service calls. Each sequence can be set as follows:

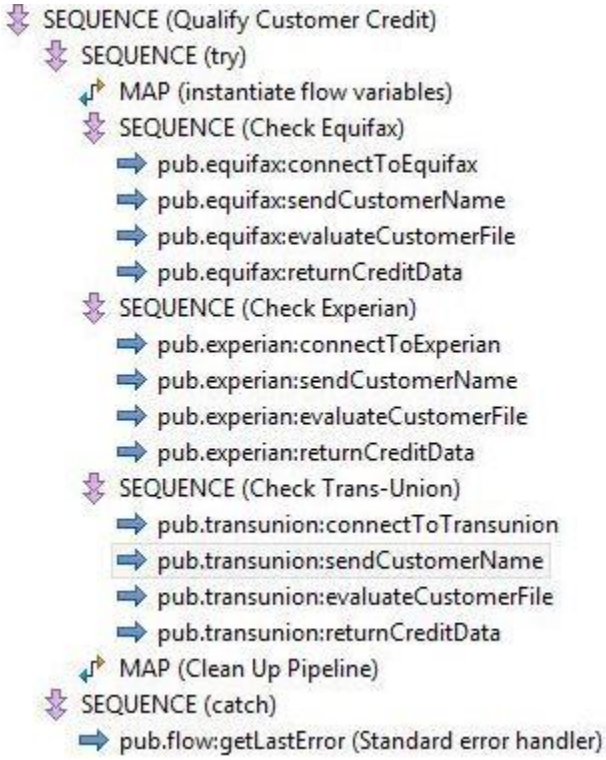
Exit Setting	Comments
Success	Typically controls groups of child Sequences. After all child sequence elements are complete, the Flow service stops. This enables try/catch.
Failure	If any service in the sequence fails, the Flow exits from the sequence and continues to the next object (usually another sequence >> catch block). If the next item is not a sequence, it executes against the object.
Done	Executes every Flow step in the sequence and does not consider errors. If an error occurs, the execution continues. This describes the catch block, however, a sequence can be used to group attempts to connect and execute services against different data sources. Another use case calls for inserting data into a database. If the database is unavailable, the data is output as an xml file and written to the file system. Once the DB is available, the file can be recalled and the data inserted. There are many ways to design and commit/rollback scenarios using simple Flow services.

The following graphic displays a multi-sequence Flow service collapse:



The try/catch blocks as well as the parent sequence are set up just as described in the previous section. The three Sequences under the try sequence, Check Equifax, Check Experian, and Check Trans-Union groups code specific to each business case. The Sequence should code that is unique and should be grouped together.

The following graphic displays the same multi-sequence Flow service expanded:



Each sequence groups specific services related to each credit bureau. Depending on the sequence Exit setting, the following may occur:

Exit Setting	Comments
Success	Will create an endless Loop until successful unless other code is introduced to exit out of the sequence.
Failure	Will exit out of the sequence and drop to the next sequence. In a try/catch block, it will execute error handling.
Done	Executes every Flow step in the sequence and but does not consider errors. In this case, it would cause issues since the sequence of Flow services are dependent on one another.

8.3 Loop Step

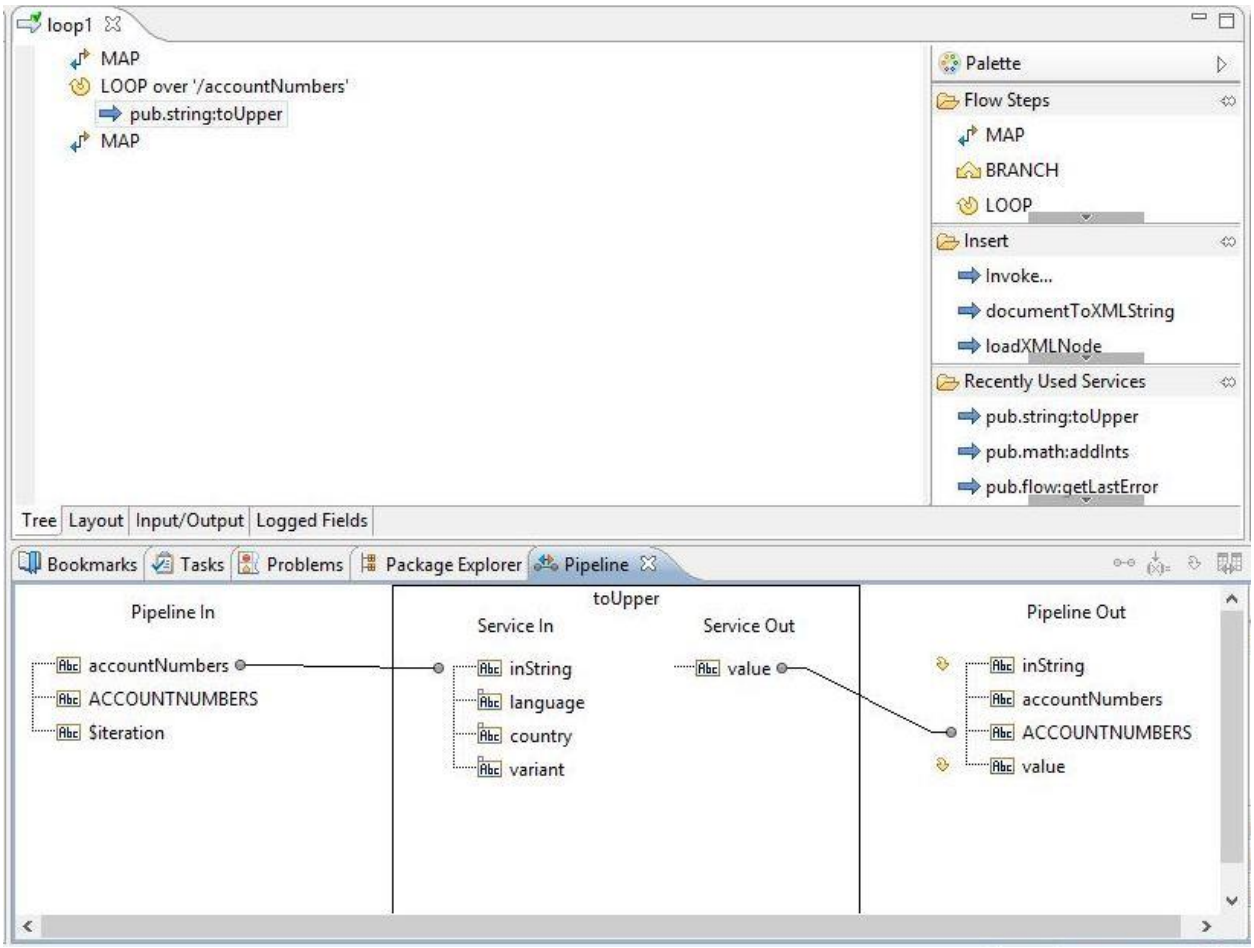
Loop steps exist to iterate through array objects. It must be used when iterating through a document structure that contains document lists. Special care in constructing the loops and the variables used will assist in maintaining application environment integrity. Careless construction or use of variables as well as not dropping unused variables during looping can cause out-of-memory circumstances. Integration Server documents can contain multiple levels or iterative data. Loop steps can contain child Loop steps.

8.3.1 One for One Loop Step Iterations

The majority of Loop Steps iterate through the entire number of array items. Loop Steps provide the capability to iterate through data, perform business rule validation and make data conversions.

The Loop Step enables code to evaluate each item in an array starting with the first item and iterating through each array item until the array index length is reached and the Loop Step completes.

The following graphic depicts a simple "1 for 1" Loop iteration that takes an account number and forces all the characters to upper case and then maps to another array:



The input array value for the Loop property is "accountNumbers" and the output array value is "ACCOUNTNUMBERS". The first map establishes pipeline variables and the last map drops unnecessary pipeline variables.

The input array cannot be modified. It is read only. Only the output array can be modified.

Tips and Tricks – 1 for 1 Loops

- Use a map to map the Array Object to a temporary variable. Set the Loop Step input array to the temporary array variable name. This will reduce memory use that could occur if large document contents are forced into the iterator.
- Use a map step prior to the Loop Step to establish variables specific to the Loop Step. Use a map step directly after the Loop Step to drop unneeded pipeline variables.
- Use a branch step and an exit step during Loop iterations to exit the Loop in the event that data fails a business rule check.

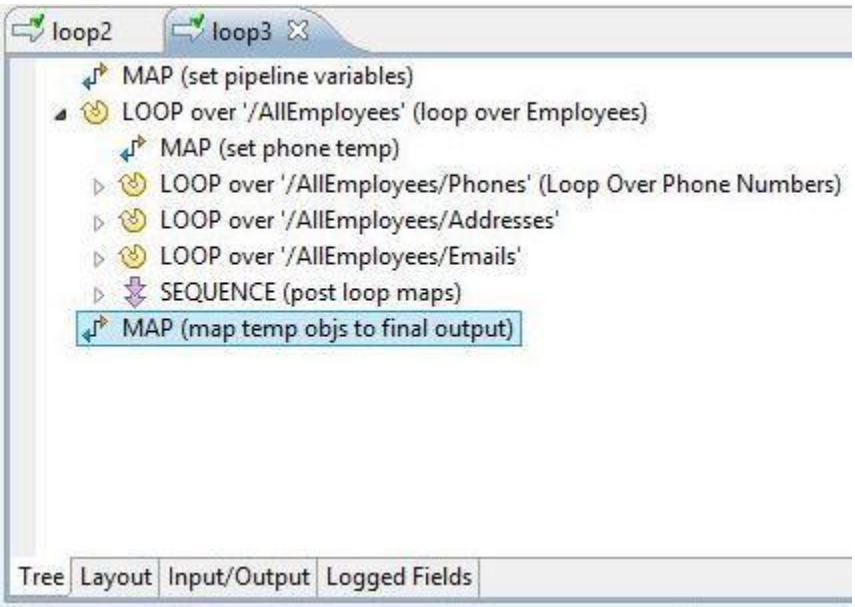
- Use the $Loop value if needed for logging. The $Loop value is the index array identifier plus 1. Array in webMethods are Java zero-based. The $Loop value starts with 1 and increments in by 1 for each iteration. This can be valuable to help debug very large data sets.
- Use the SCOPE property to limit the amount of data introduced into the Loop Step.

8.3.2 Narrow Down Data Sets with Loop Step Iterations

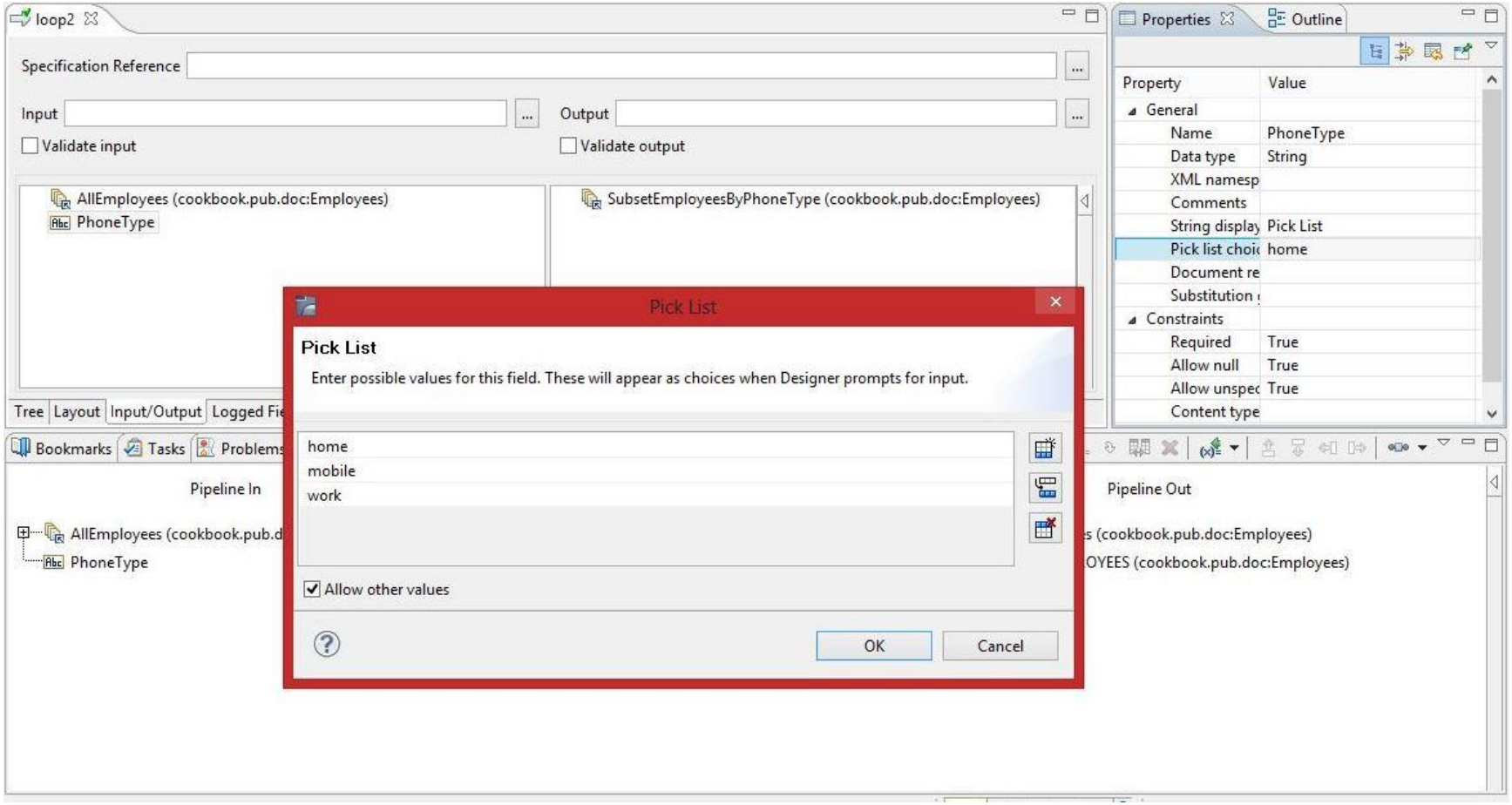
Sometimes, a need arises to evaluate array data to determine exceptions (i.e., Purchase Value > Credit Limit) or find specific records (all customers who live in the state of Maine). During each iteration, the code evaluates data and takes action. This is different from the 1 to 1 Loop Step type because it involves extracting a subset of data from the larger data set.

The next several pages contain an example of a Flow service that contains complex loops. The example uses 2 loops. The first Loop iterates through a fictional employee records. The sub-Loop of the employee record iterates through all phone numbers provided by the employee and filters out records based on the variable, "PhoneType" (home, work mobile). The Flow returns all employee records that match the selected PhoneType criteria. The input for the service is Record List Reference Object that contains "AllEmployee" data. The output (SubsetEmployeesByPhoneType) uses the same Record List Reference Object but will only contain records that meet the criteria selected.

This example is meant to provide additional insight into how to use loops. If the code were needed to iterate through all sub-record arrays (phones, addresses or emails), then the Flow would resemble:

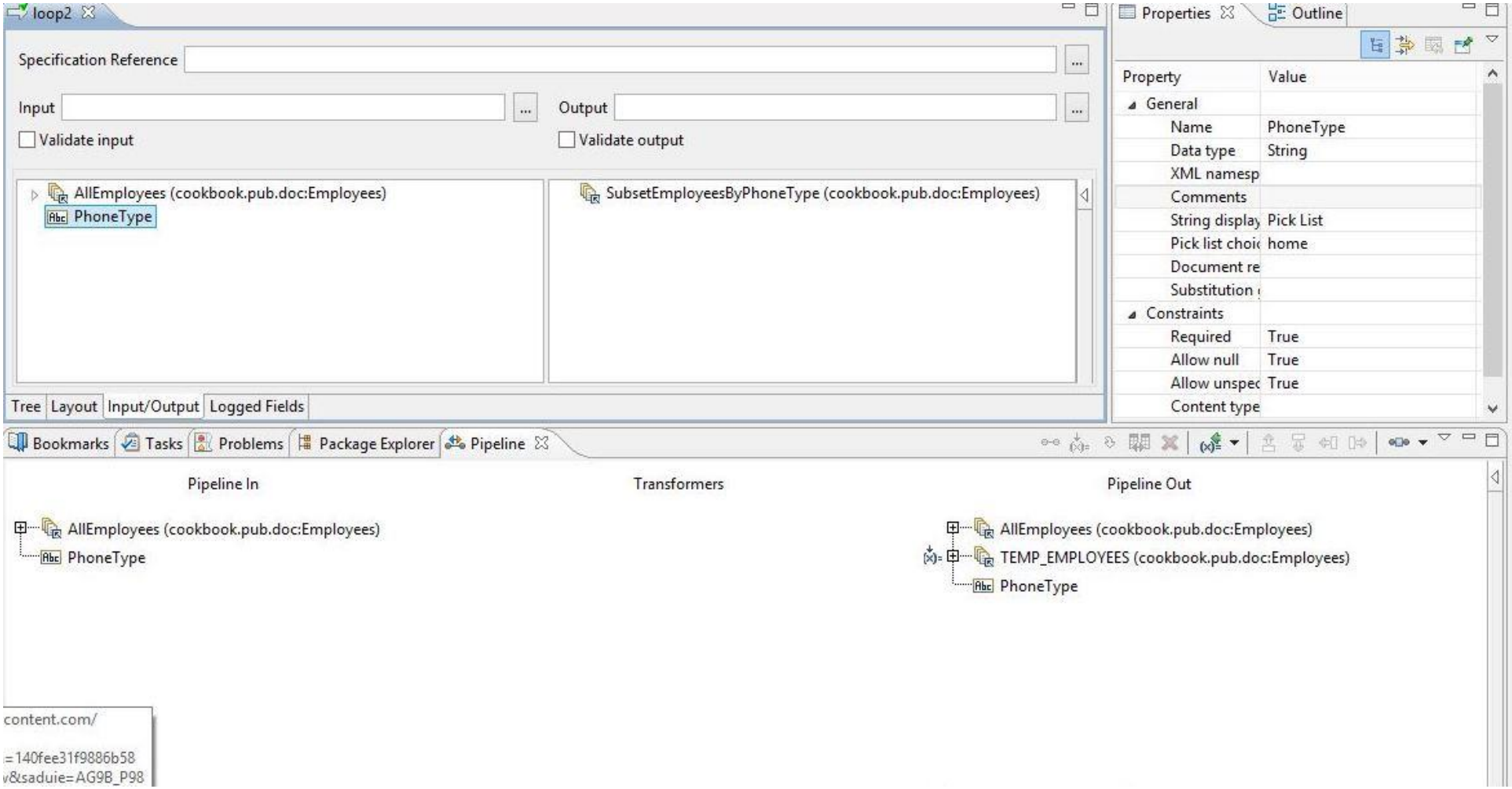


The example only iterates through the phone records for each employee.

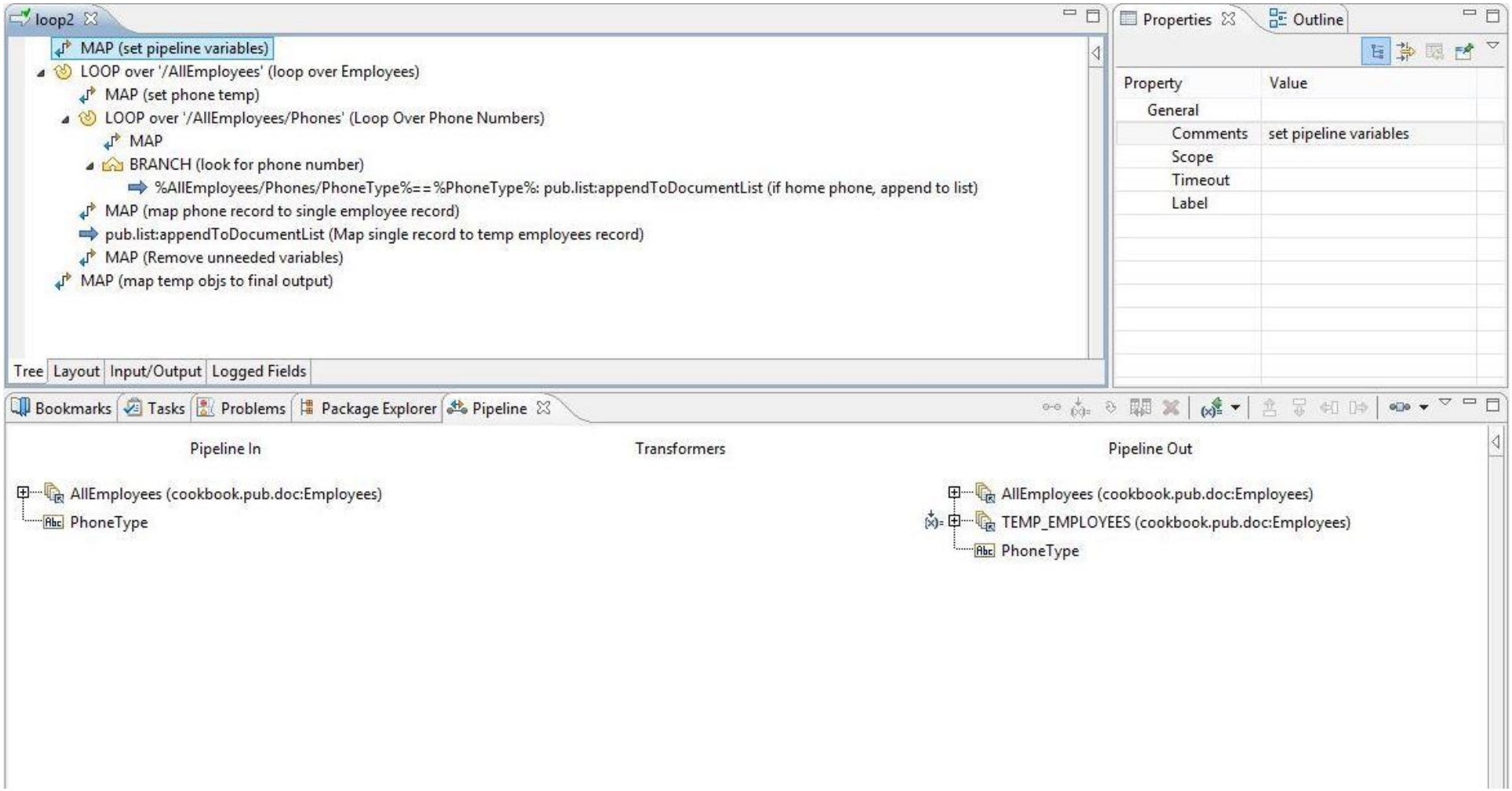


The input signature includes a document reference list that contains Employee information and the "PhoneType" variable (pick list choices: home, mobile, work.) To view this information, make sure the target Flow is selected in service development view and the input/output tab is selected.

The properties tab contains 2 differences: String display is set to Pick List. The pick list is built using the pick list choice button in the properties tab in the value box. Notice the "Allow other values" box is checked above. If the box is unchecked, the list will be restricted. Checking the box allows additional values to be input at runtime.

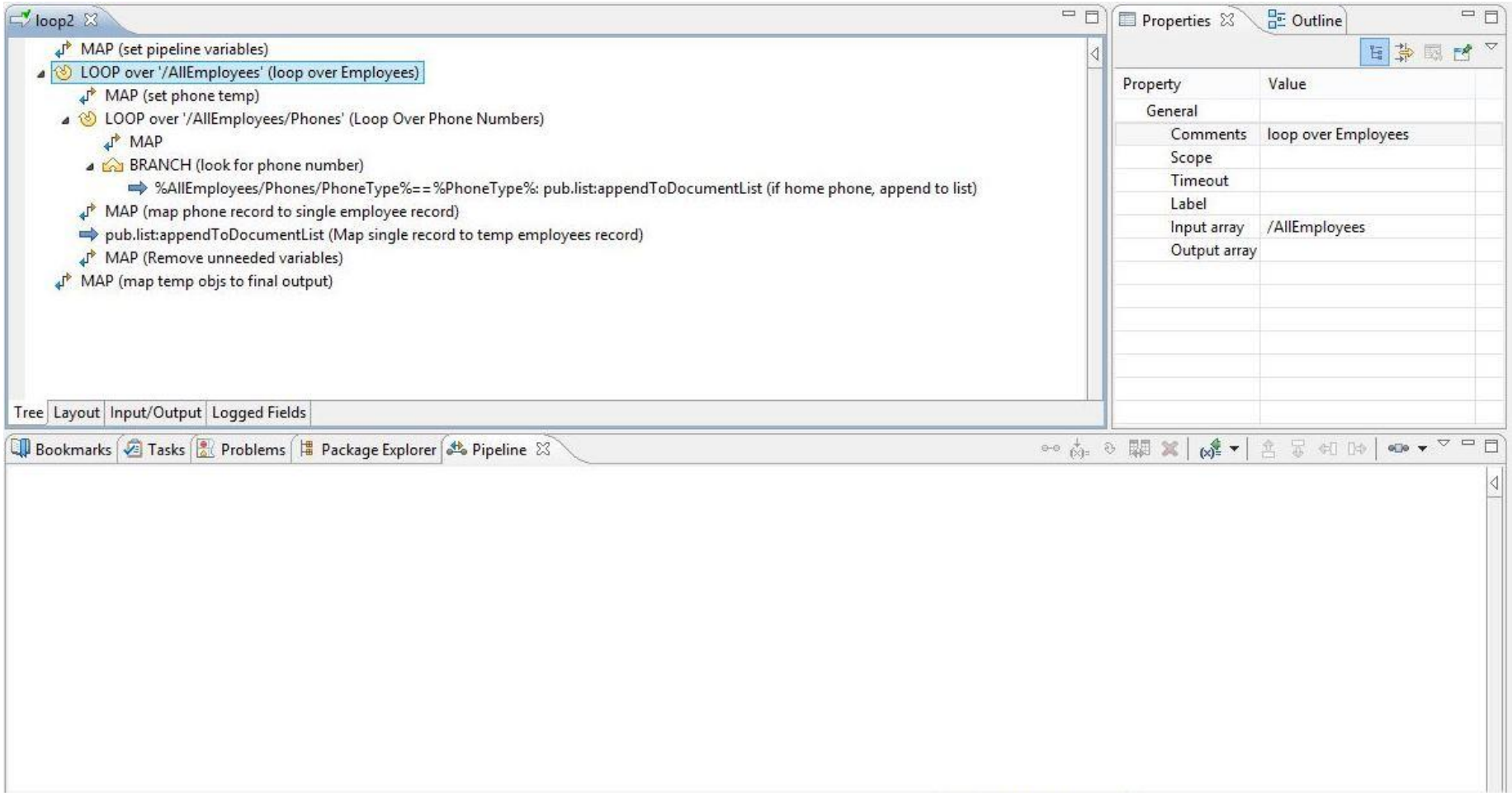


The first map step requires a separate copy of the Employees Document Reference List (TEMP_EMPLOYEES). This temporary array structure will be used to store records that meet the criteria selected at runtime. The purpose of the example Flow is to identify all employee records that meet a particular criteria. To obtain a list of employee names with HOME phone numbers, choose "home" out of the 'PhoneType' Pick List choices. Running the Flow service will result in a subset of all employees who have home phone numbers within the data file. If all employees have home phone numbers, then the subset array length will be the same as the original record set. In the case where employees did not register a home phone number, the Flow service will not include their employee information in the output array. Each employee record that matches will be mapped to the TEMP_EMPLOYEES record.

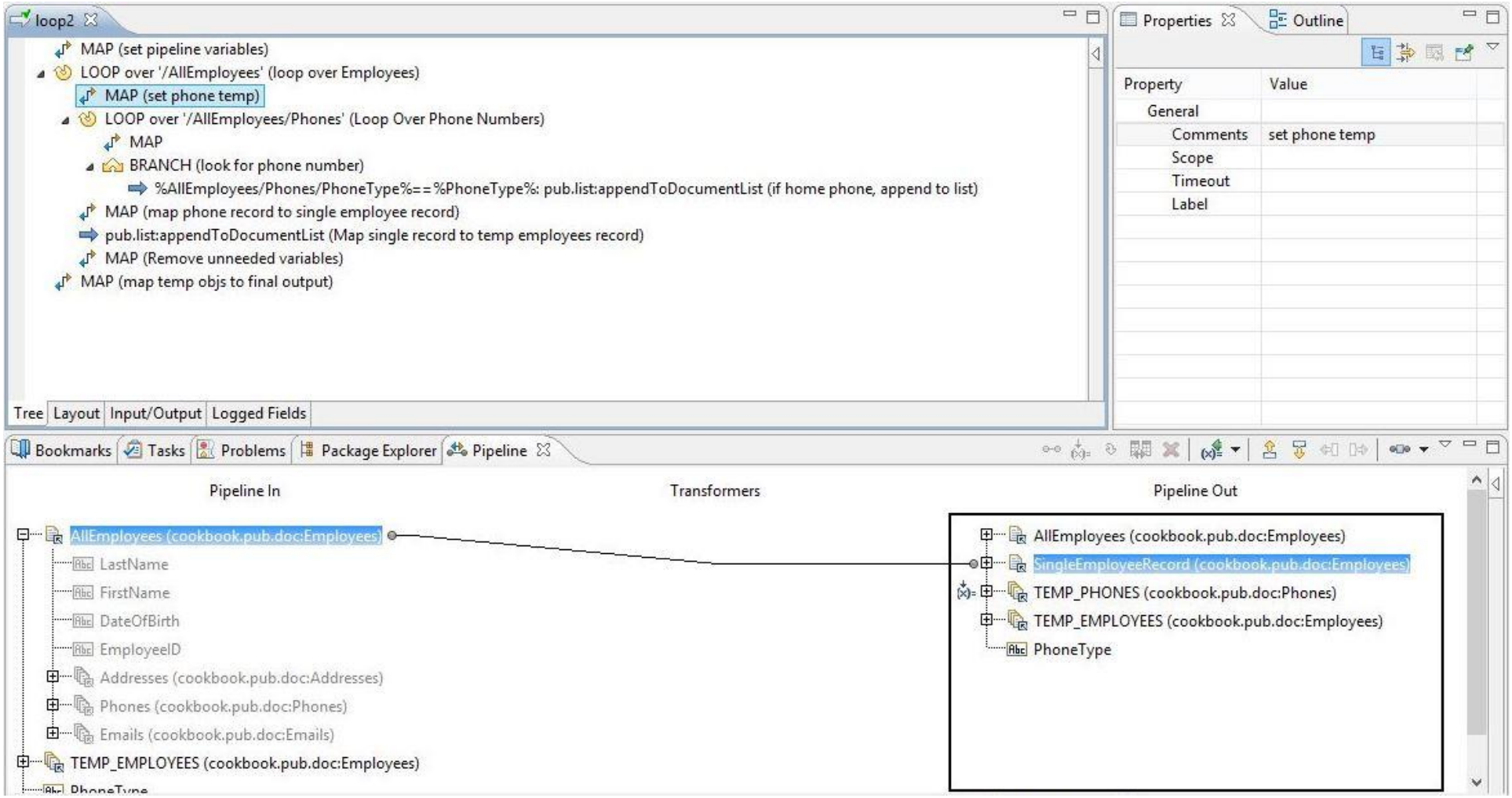


To obtain this view, select the tree view above left and make sure the pipeline tab is selected. This shows the pipeline view for the first map step, set environment variables. To make sure the correct item is selected (which will display the correct properties for the given object), match the comment field in the properties tab against the comments that can viewed in the tree view. The first map step comments are “set pipeline variables” which matches the comments listed in the property tab.

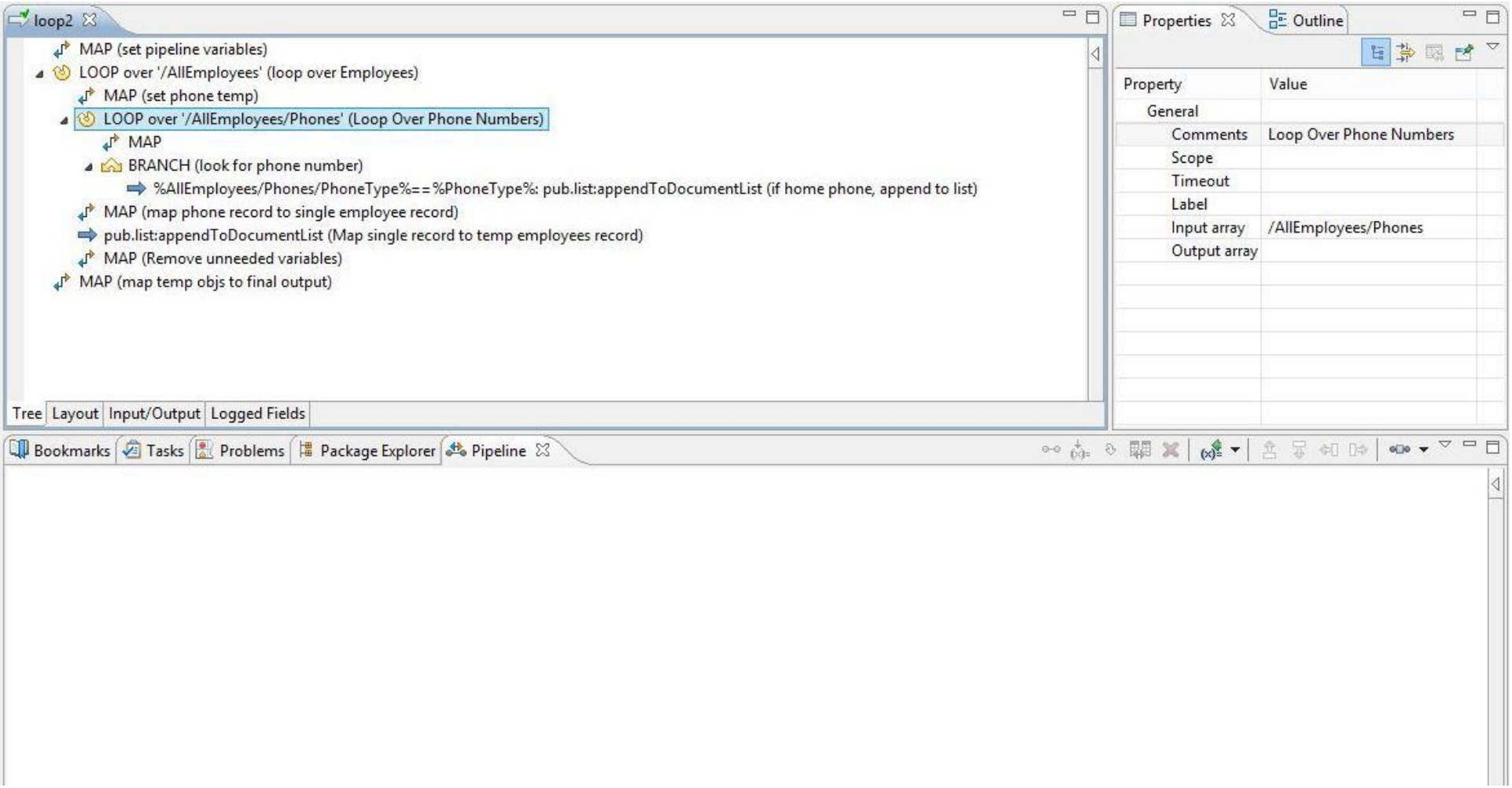
As mentioned on above, TEMP_EMPLOYEES variable is set during this initial map step.



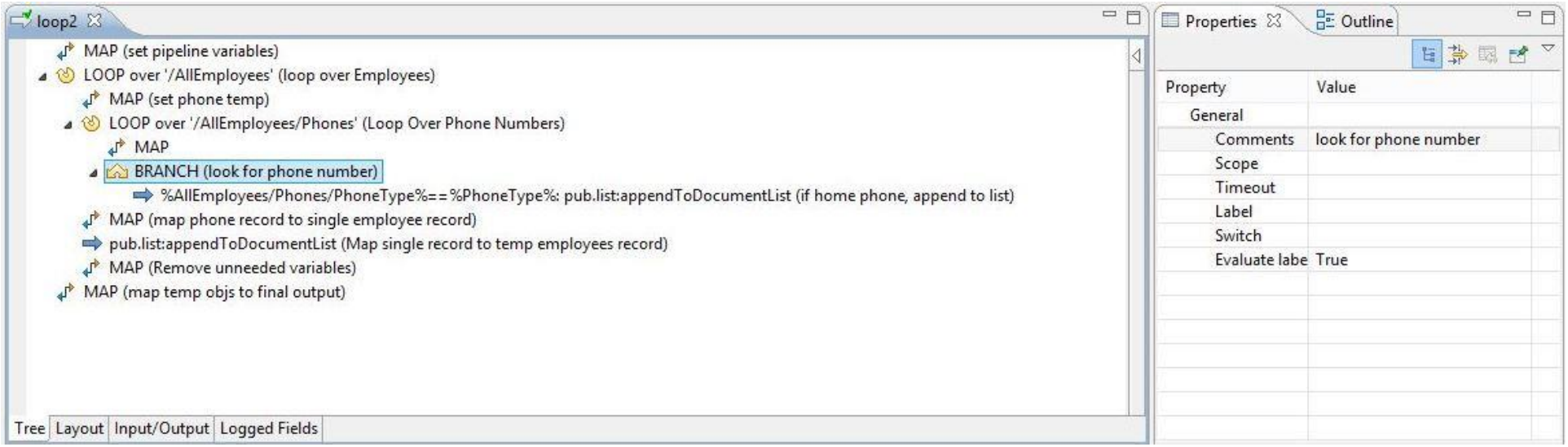
The Loop input array property is set to iterate over the all records contained in AllEmployees. Each iteration of this first Loop provides a single array index item for each record. Each employee record contains an array of phone numbers. The second Loop evaluates the phone type, and appends a sub-record with the selected phone type (home, work or mobile). The branch step evaluates each PhoneType and only appends the record containing the PhoneType selected. To make the code a little more efficient, the branch child step could have been a sequence that contains the pub.list:appendToDocumentList and an exit step (set to exit $loop) which would have exited to the higher level loop. Each employee record is appended to TEMP_EMPLOYEES and the final map step maps the data to the record indicated in the output signature, SubsetEmployeesByPhoneType.



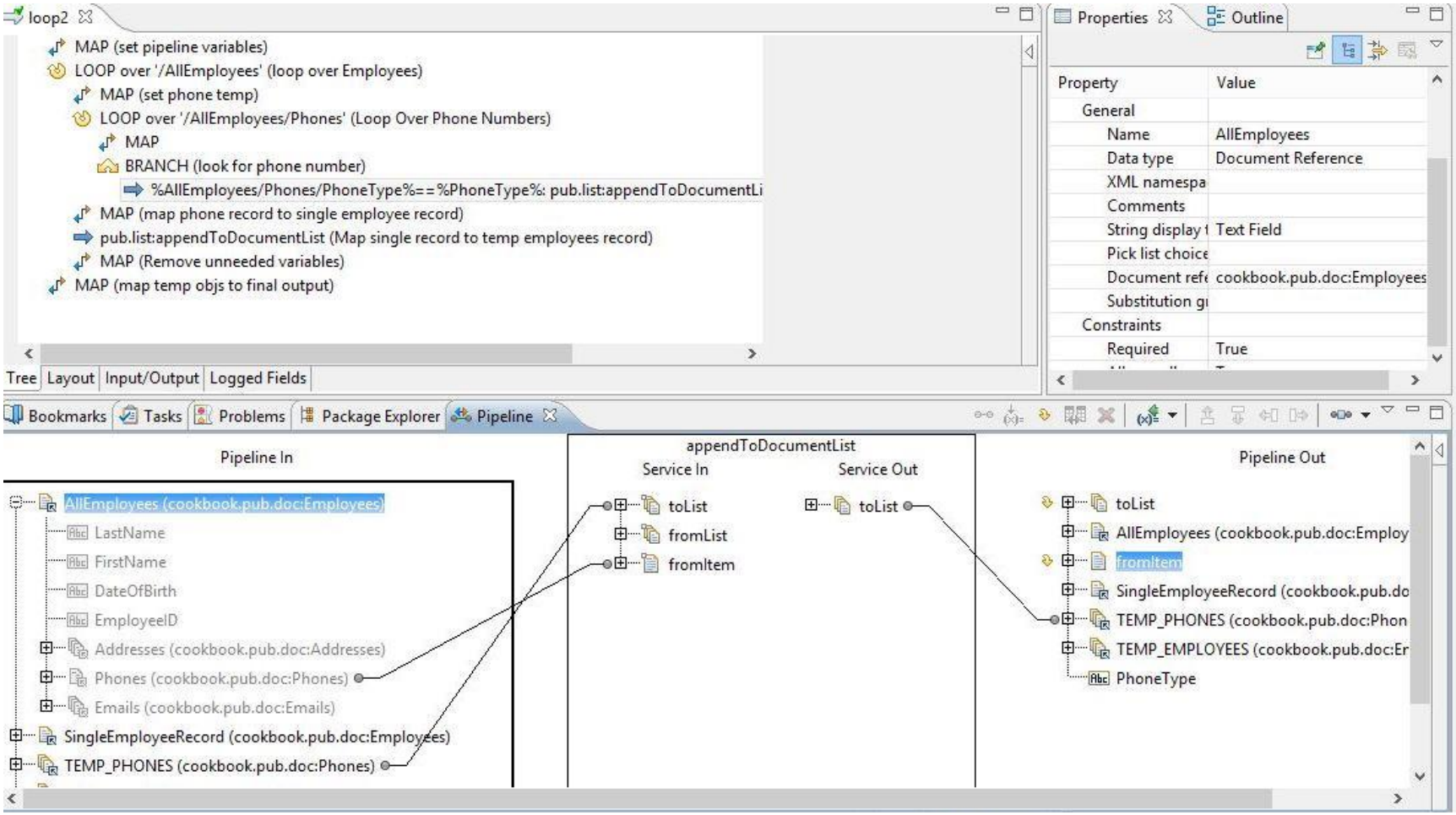
This step is important. The "AllEmployees" record is a single document reference (because it is inside the loop). This record is immutable, so to create a subset to process later, it must be mapped to a temporary single record (SingleEmployeeRecord). When this is mapped, all sub-elements of this Record List are mapped, including the Phones array.



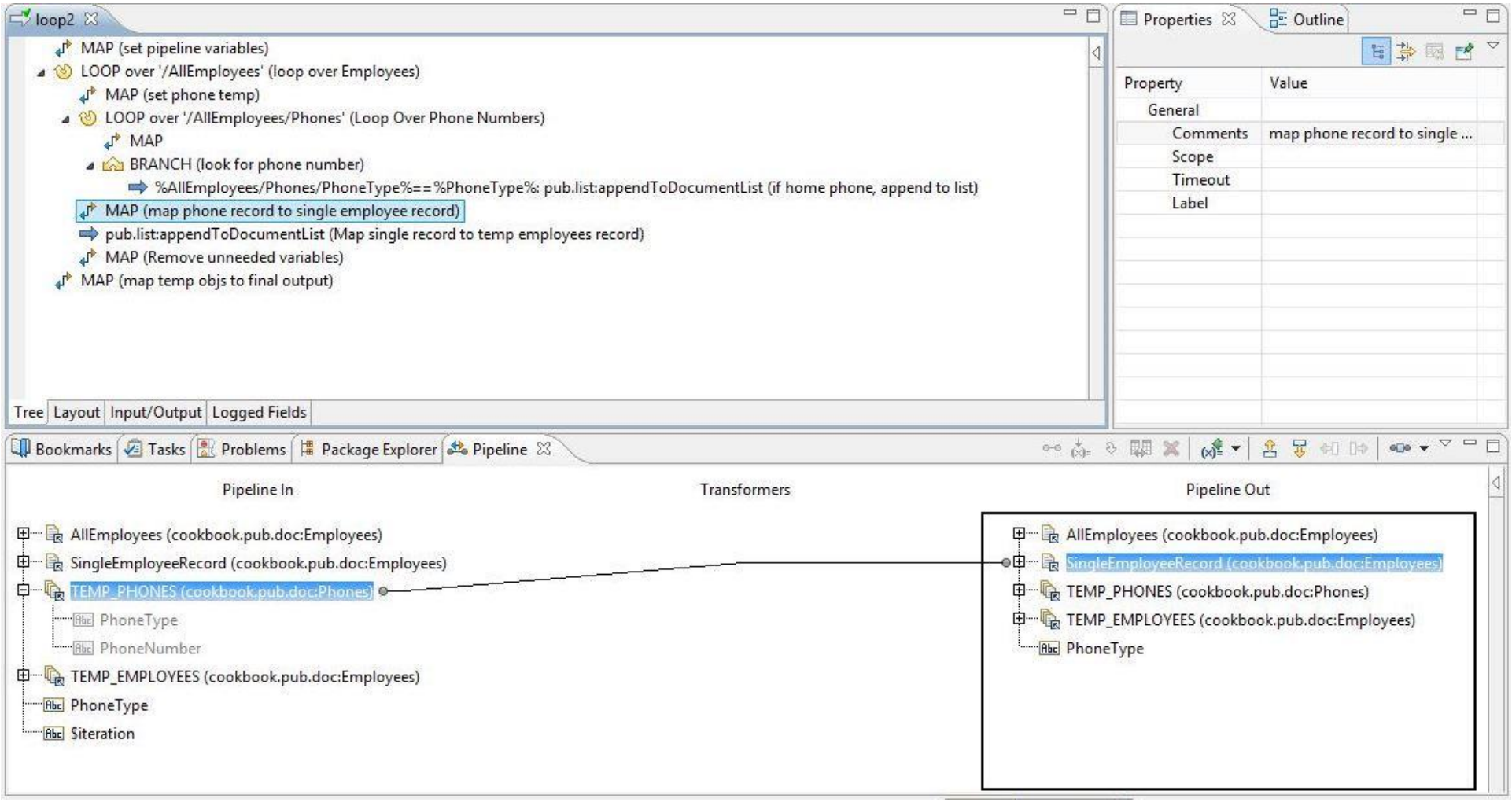
This Loop input array property is set to /AllEmployees/Phones. This will iterate over each phone record. The branch compares each record with the PhoneType and creates a temporary "Phone" record with the findings.



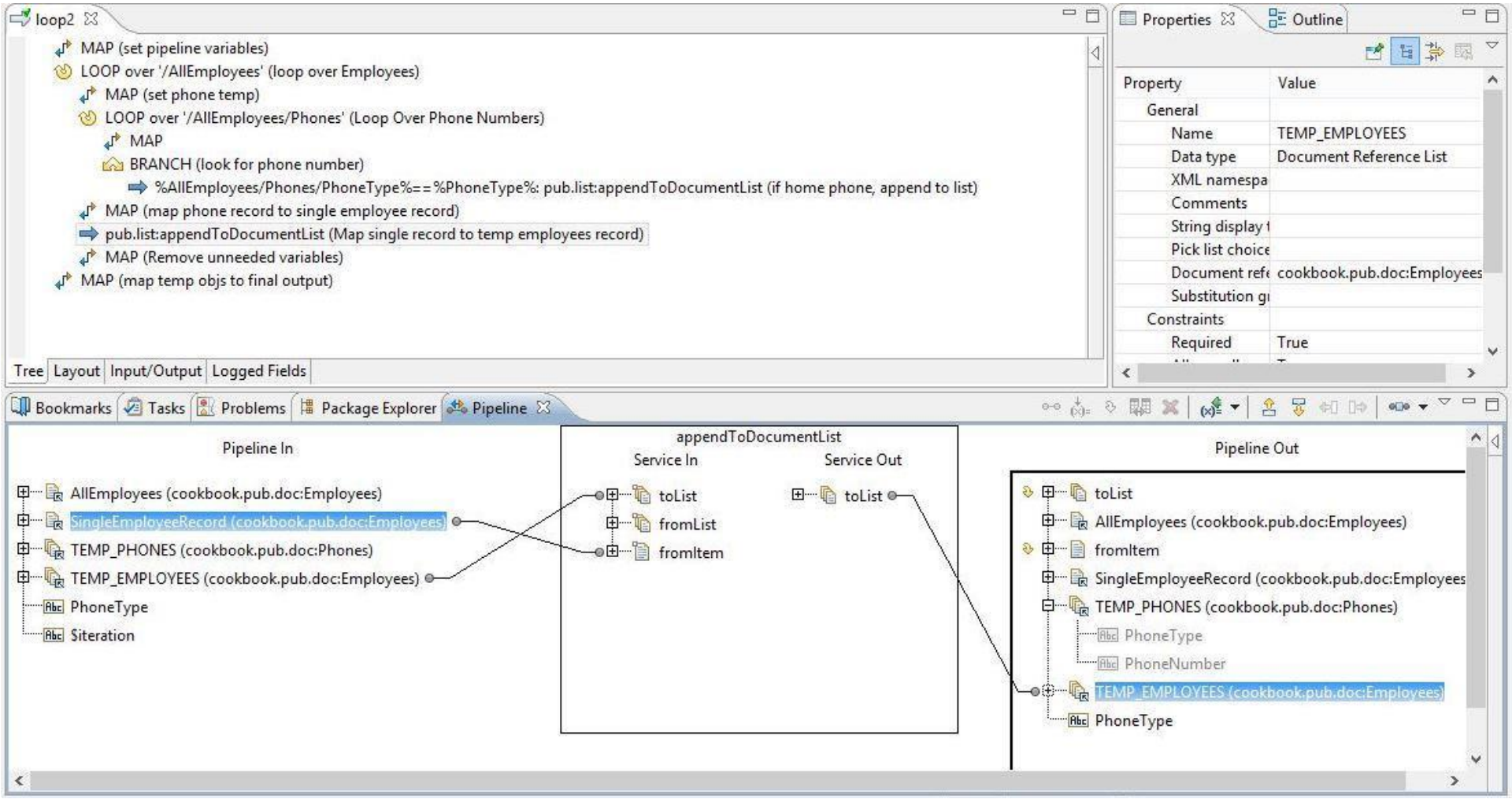
The branch statement runs during each Loop iteration. Notice, that the evaluate labels property is set to "true". In this case, for each phone record, where "/AllEmployees/Phones/PhoneType" is equal to the input "PhoneType", the Flow service under the branch will execute. It will not execute unless the PhoneType matches. This simple example is not optimized but is a demonstration of nested loops.



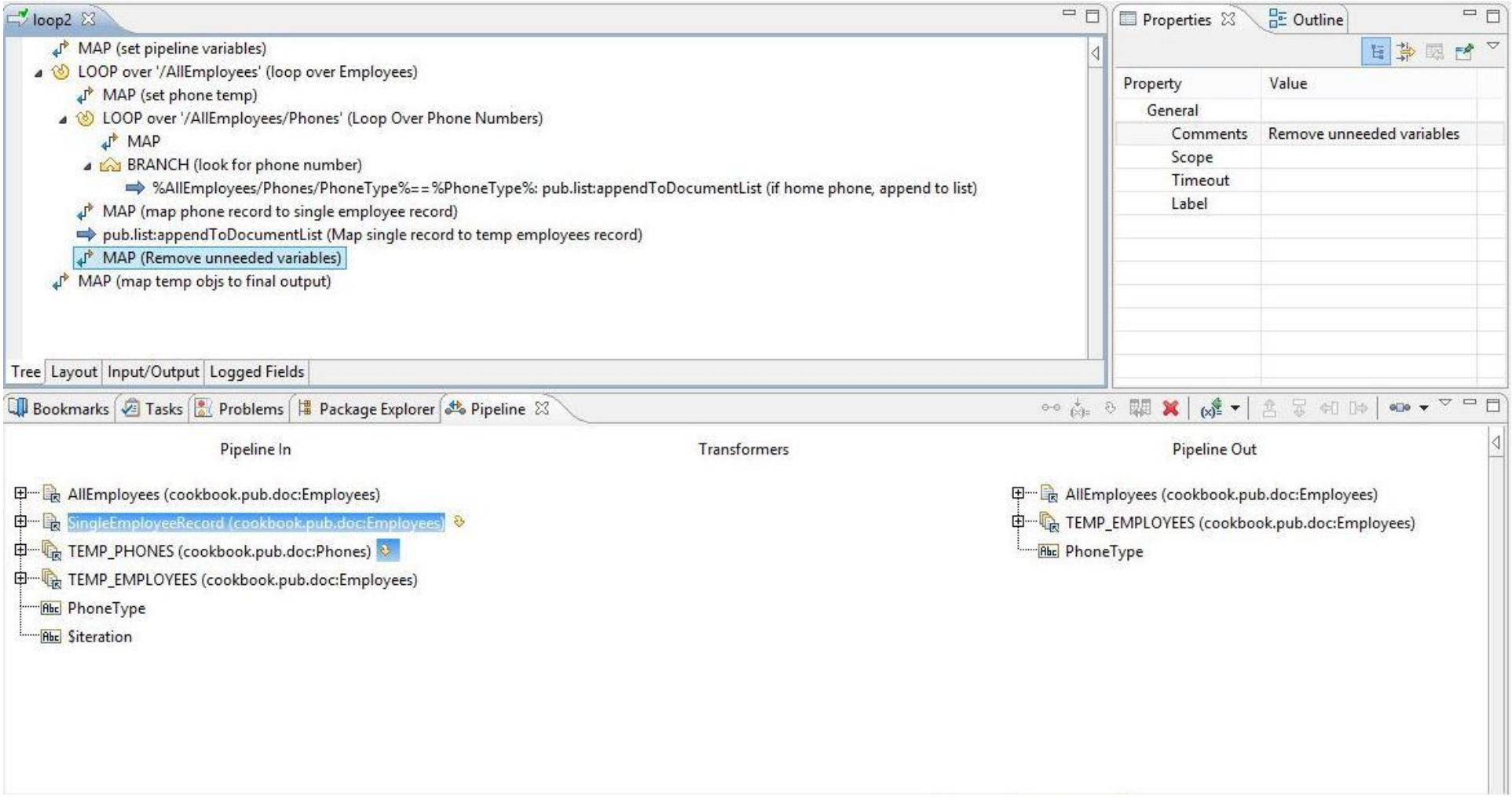
Each record that matches the "PhoneType" criteria will get appended to the "TEMP_PHONES" record. The utility Flow, appendToDocumentList, collects each record that matches PhoneType into a temporary array for later mapping purposes. Notice that the sub-array "Phones" under "AllEmployees" in the Pipeline In the column on the left is a single document instance. This verifies that the array is in a Loop iteration in the Flow step. For this scenario, the TEMP_PHONES record is mapped to the "toList" Service In variable, and the "/AllEmployees/Phones" record is mapped into the fromItem variable. For the first iteration, the TEMP_PHONES array will be null. After the append, it will contain the record from "/AllEmployees/Phones". This is repeated for all records.



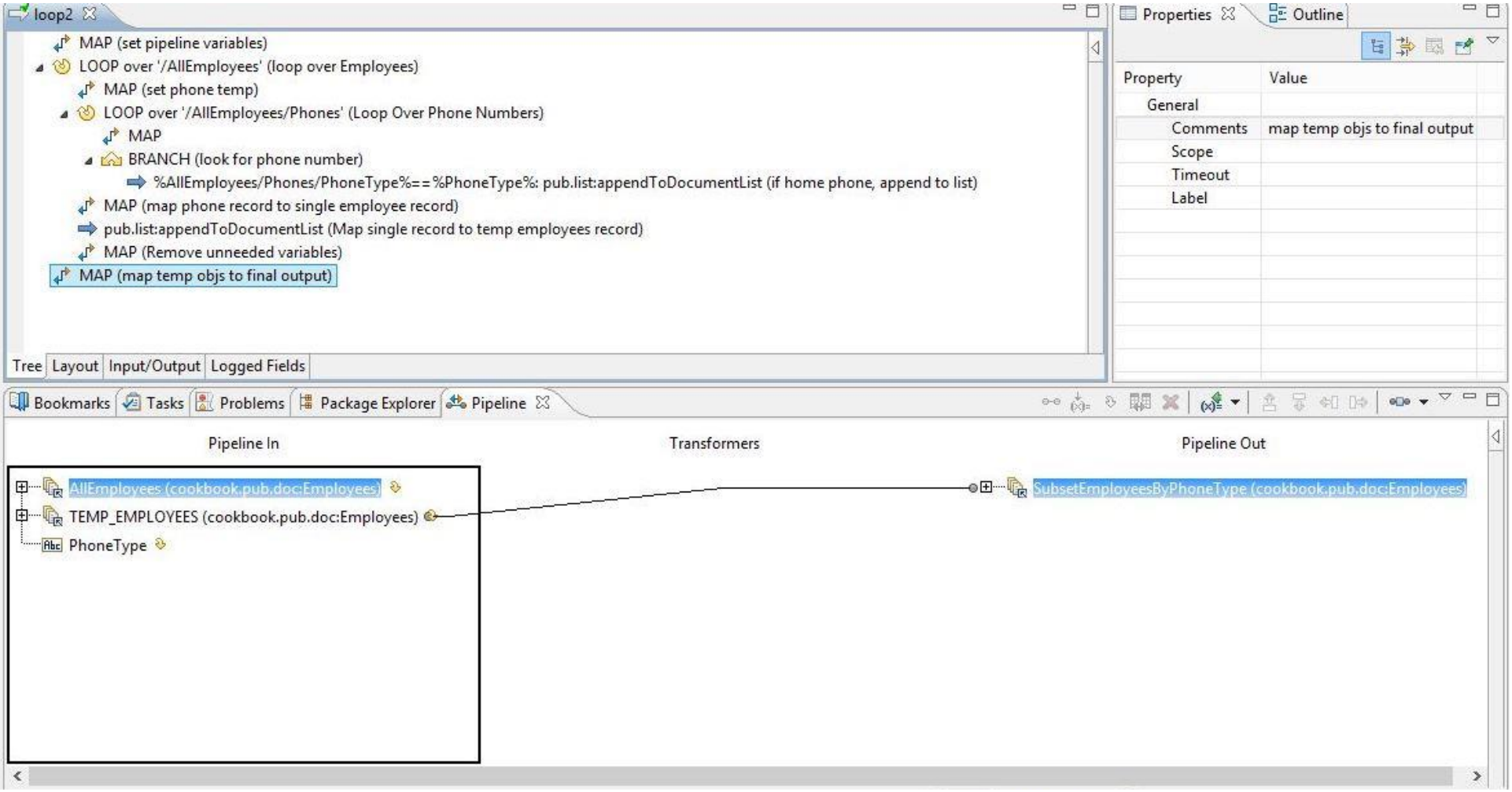
The preferred mapping pattern is referred to as the "Donut" hole. Always work from the inside to the outside of a data set. In this case, the Phones array is inside so it is mapped to the temporary record, "SingleEmployeeRecord". Recall that the SingleEmployeeRecord is mutable, in that it may be changed. The original Phones sub-array is overwritten with just the selected Phone record. This represents a single line record of an employee with a selected PhoneType.



This step appends the "SingleEmployeeRecord" data into the TEMP_EMPLOYEES record list (that was initialized in the first map step of the Flow). The TEMP_EMPLOYEES record list will be null until the first append. Each record will be appended. (See end of this section for code optimization.)



This step removes unneeded variables. Note the "$iteration" variable. This variable contains a number that is equal to the Iteration of the loop. The first array item, 0, is $iteration, 1. This is the last step of the "/AllEmployees/Phones" sub-loop. It will repeat until the outer, main Loop is complete.



This is the last step in the Flow service. It drops all pipeline objects and maps the temporary objects to the final object specified in the output signature. This Flow highlights mapping patterns and the use of temporary variables. The structure of the Flow (without try/catch blocks) suggests that this could be a child Flow called by a parent Flow service that provides for error handling.

Map Steps and Optimizations for Aggregate Loop Data. The typical steps to create a Loop that filters and aggregates data into a subset include:

1. Use a map step before each Loop to create an appropriate temporary object variable to map data. The iterator uses the original object and the Flow will require a new object to map information/data into.
2. Create a map step as the first child step of the Loop to create and initialize variables. They are used for each iteration and are dropped during the last map step for each Loop iteration.
3. Use a sequence to group evaluation logic as a child step of the Loop if appropriate. The idea is to evaluate the records, isolate, and append to a new record that is in the output signature for the Flow.
4. For each record that meets the criteria, use pub.list:appendToDocumentList or a similar service to append the target data set to object variable created in the map step prior to the Loop step. This will create and append a smaller subset of records. The Loop Step is complete when the last iteration occurs or if the code uses an Exit Step.
5. This same technique can be used to scan data for exceptions. An exit step is used to exit the Loop after criteria is met.
6. Always use a map step after the Loop step to drop unnecessary pipeline variables.

Tips and Tricks – Complex Loops

- Use the first map step to set Boolean variables. These variables should be set to **false**. After a condition is met in a branch statement, set the variable to **true**. The value can be used decide when to append data to a record.
- Use temporary variables inside loops and drop them when the Loop is finished.
- Call these Flow services from a parent and use the SCOPE property to limit the amount of data provided.
- To sample a small set of data, set a branch statement to evaluate the $iteration value. Set the evaluation to the number of records desired. For example, the branch statement (set evaluate labels to true) could check if $iteration == 100. At record 100, execute an exit step that exits $Loop (retaining records).

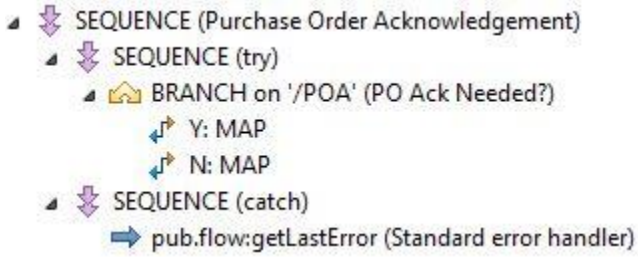
8.4 Branch Step

Branches correspond to If/Then or case statements. There are two ways to utilize branches: By evaluating labels or by evaluating a specific value in the pipeline.

8.4.1 Evaluating Labels

Evaluating labels provides the ability to compare the value of several pipeline variables (adding, subtracting, BOOLEAN, etc.) but assumes the existence of the variables. Special care should be used to check for the existence of variables or else a runtime exception may be thrown. Be using the default setting for Branches, the Branch statement corresponds to an If/Then statement. Each sub-object under the branch is given a static value (like Yes or No). So, if you are checking to see if a

Purchase Order Response is needed for a customer, you could examine a pipeline variable, "POA". This is handled by setting the branch to "**evaluate label = false" and switching on the pipeline variable, "POA".** Here is a code example:



If the value in the pipeline for POA = "Y", then the first map is chosen, and if POA = "N" the second map will be used. If the value of POA is anything other than "Y" or "N", then nothing occurs for the POA. The following table depicts the required Branch property settings:

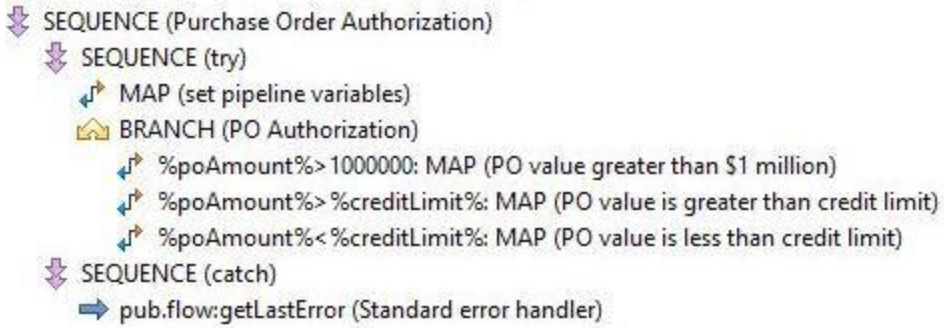
Property Setting	**Property Value**
Comments	PO Ack Needed
Scope	Leave blank
Timeout	Leave blank
Label	Leave blank
Switch	POA
Evaluate Labels [true | false]	false

In the label property for each child map, enter in the value. In the graphic above, the first map step will contain a label property value of "Y" and the second map step will contain the value of "N". These property values are the possible values for the pipeline variable "POA". Any other value will not be evaluated.

8.4.2 Evaluating Pipeline Values

This leads to the other possible BRANCH STEP use case. This is enabled by setting the Evaluate Labels property to true.

These Branches should be used when a pipeline value can have several different values, or when the value is not known and a mathematical or logical comparison must be made prior to continuing in the Flow. Branches should be used to evaluate or confirm business logic and to throw errors in conjunction with an exit process. Setting the "Evaluate Labels" property to "true" requires a different configuration for the underlying maps or Flow services choices under the branch.



The inputs to this Flow service are "poAmount" (amount of purchase order) and "creditLimit" (the total amount of credit permitted). The output is "creditApproved" (Boolean). The business case statements for Purchase Order Authorization include:

1. If poAmount is greater than $1 million, creditApproved == false.

   %poAmount%>1000000
2. If poAmount is greater than creditLimit, creditApproved == false.

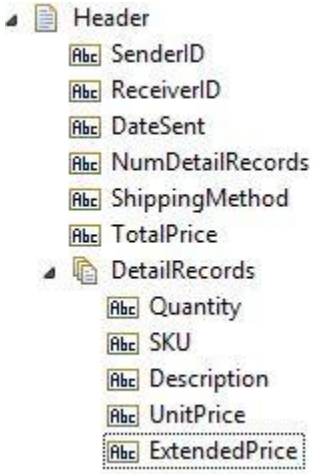
   %poAmount%>%creditLimit%
3. If poAmount is less than creditLimit, creditApproved == true.

   %poAmount%<%creditLimit%

Set the property, Evaluate labels ==true, if data can be dynamically applied against a formula or business rules.

A few basic rules need to be followed:

1. All pipeline variables must be wrapped with percent signs. A variable named "MyInput" would resemble: %MyInput%. It is important to match case since the underlying technology is Java.
2. A simple way to be sure that the variable name is correct is to copy the element and paste it into the map or Flow step. Examine the following Document:



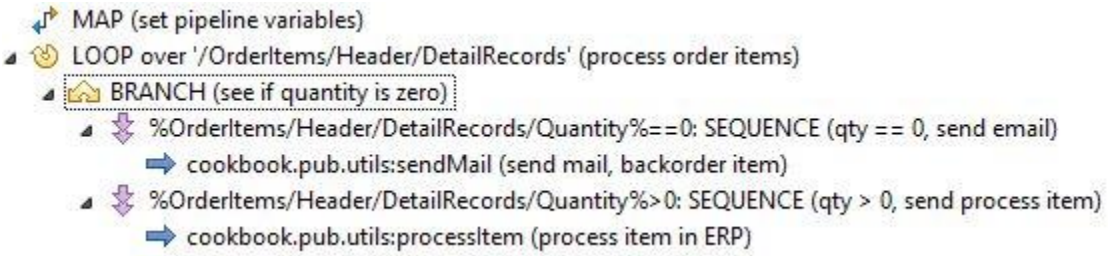
Selecting "ExtendedPrice", and copying and pasting it into a pair of "%%", would result in: %Header/DetailRecords[0]/ExtendedPrice%

Notice that "DetailRecords" has an array index assigned. This is because it was copied from the Document Element. To use this variable in a statement, remove the array index "[0]". The variable will resemble: %Header/DetailRecords/ExtendedPrice%. This step can be avoided if the copy and paste is conducted in a child map of a Loop or Repeat step.

3. A Loop step or Repeat step can use a BRANCH Step (evaluate labels set to true) to evaluate if a given condition is true, false, or if it fits into a mathematically correct formula. Under the branch, sequence steps are frequently used (so multiple steps can fire off). Using the previous graphic as an example, a Loop could evaluate if the quantity of an item is zero:

%Header/DetailRecords/Quantity%==0.

In this case, the child steps of the sequence step could send an email to the Sender indicating that this item is not in stock and will be backordered. The following image depicts the Flow:



The following table depicts the required for the Branch property settings:

Property Setting	**Property Value**
Comments	PO Authorization
Scope	Leave blank
Timeout	Leave blank
Label	Leave blank
Switch	Leave blank
Evaluate Labels [true | false]	True

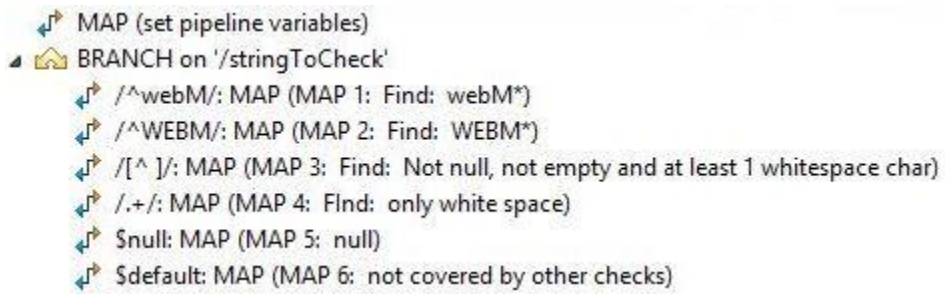
8.4.3 Using Regular Expressions (REGEX) to Evaluate Branches

Regular expressions are used across all modern programming and scripting languages. webMethods Integration Server provides support for Regex using the Branch Step. Evaluate labels should be set to true and the REGEX expression inserted into the child object's label property (Map, Sequence, Flow, etc.)

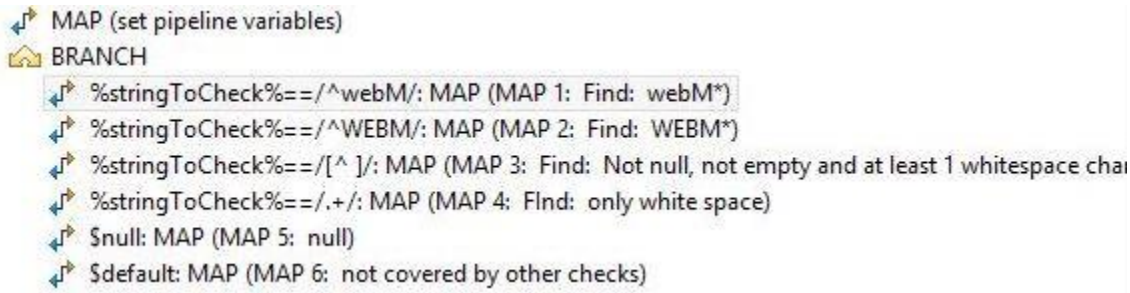
The following six test conditions are reflected in the Regex code with an input string variable, "checkForString", and an output string variable, "stringOutput". The output variable, for this test, indicates which map was selected by the code. From top to bottom, they are MAP 1 to MAP6.

Test	checkForString	Map Used	Expected Result
/^webM/	webMethods	MAP 1	Returns if input == webM*
/^WEBM/	WEBMETHODS	MAP 2	Returns if input == WEBM*
/[^]/	Webmethods	MAP 3	Returns if chars <> MAP 1 or MAP 2, not null. CAN ENTER ANY STRING.
/.+/		MAP 4	Enter white space only (use space bar)
$null		MAP 5	If null
$default		MAP 6	Not covered by the first 5 maps

In the case of Evaluate labels, false, the branch is set up normally. Each child step of the Branch Step contains a possible value but uses REGEX for evaluation.



In the case of Evaluate labels, true, the branch is set up normally. Each child step of the Branch Step contains variables contained within percent "%" signs along with an operator.



Either Branch Step approach will work but the second approach allows for the chaining of values. For example, you can use operators to evaluate multiple pipeline values and use "and", "either/or", types of operations.

The following table provides a brief overview of standard relational operators that may be used in Branch Steps:

STANDARD RELATIONAL OPERATORS

Operator	Syntax	Use	Code Example
=	a = b	Equal to.	%CompanyName%='Boeing' %boolOutput%='true' In string comparisons, can use single tick or double tick: ' or "
==	a == b	Equal to.	%CompanyName%=='Boeing' %customerState%=='MD' In string comparisons, can use single tick or double tick: ' or "
!=	a!=b	Not equal to.	%totalPrice%!=%totalDetailLines% Total price should the total of all detail lines added together. This
<>	a<>b	Not equal to.	%totalPrice%<>%totalDetailLines% Note: Total price should be the total of all detail lines added together.
>	a>b	Greater than.	%totalPrice%>%totalDetailLines% Note: Total price should be the total of all detail lines added together. This implies that not all detail lines were included in the submission.
>=	a>=b	Greater than or equal to.	%creditLimit%>=%purchaseAmount%
<	a<b	Less than.	%purchaseAmount%<%creditLimit%
<=	a<=b	Less than or equal to.	%purchaseAmount%<=100000 Note: To evaluate numbers, DO NOT USE string qualifiers of: ' or "

The following table provides a brief overview of logical operators that may be used in expressions:

LOGICAL OPERATORS

Operator	Syntax	Use	Code Example
!	!*expr*	Not	Negates the next expression ! (%state% == 'CA')
not	not *expr*	Not	not (%taxable% == "true")
|	*expr* | *expr*	Or	%day% == 'Saturday' | %day%=='Sunday'
||	*expr* || *expr*	Or	%day% == 'Saturday' || %day%=='Sunday' %day% == ('Saturday' || 'Sunday') %customerType% == ('MILITARY' || 'Military' || 'military')
or	*expr* or *expr*	Or	%day% == 'Saturday' or %day%=='Sunday'
&	*expr* & *expr*	And	%discountAmt% ==10 & %customerType%=='military'
&&	*expr* && *expr*	And	%discountAmt% ==10 && %customerType%=='military'
and	*expr* and *expr*	And	%discountAmt% ==10 and %customerType%=='military'

Of great importance is the precedence of the operators. The following table notes the operator precedence level and helps to define which should be used in a given circumstance.

Precedence Level	Operators
1	( )
2	not, !
3	=, ==, !=, <>, >, <, >=, <=
4	and, &, &&
5	or, |, ||

To override the order in which expressions in a condition are evaluated, enclosed the operations first in parentheses. Integration Server will evaluate expressions contained in parentheses first.

Tips and Tricks – Branches

- Set evaluate labels to **false** when the evaluated pipeline data is static or the return set is limited
- Set evaluate labels to **true** when there is a need to evaluate variable data or use formulas.
- Formulas can evaluate numeric data, dates, Strings.
- Cannot be used to evaluate the content of objects such as byte arrays, streams, etc.
- Can be used to evaluate $null for objects (i.e., %bytearray% <> null)
- Any object can be executed under a branch: maps, Flows, other branches, Sequences, loops, etc.
- Regular Expressions can be used when evaluate labels property is set to **true**.

8.5 Repeat Step

The Repeat step operates like a Loop in that the underlying structure is a "while" statement. Remember, loops only iterate through elements of an array start with index[0] and ending with index[].length();. The Repeat step repeats a series of child operations. At design time, the developer configures the Repeat Step to repeat based on whether the operation fails or succeeds (failure or success). The statement will continue to repeat 'n' times until the count is reached. The repeat function does not count the first iteration. To repeat activities 10 times, the count should be set to 9.

The following table contains the most common types of repeat-based Flow services:

Failure	Success
• Database connection retries • System connections whether messaging, JMS, ftp, etc. • Could be set indefinite or for a set number; escalate a notification.	• Large file parsing; continue to read file until there is no more bytes to read which causes failure (not success). • Text searching. Not the most efficient, but a quick method to search for a string. Repeats on success, ends on failure.

Connecting to other resources like databases, web services or other TCP/IP make good candidates for Repeat Steps that repeat on failure. The repeat will either indefinitely continue to retry the connection until it successfully connects or until it meets the repeat count specified in the Repeat Step properties.

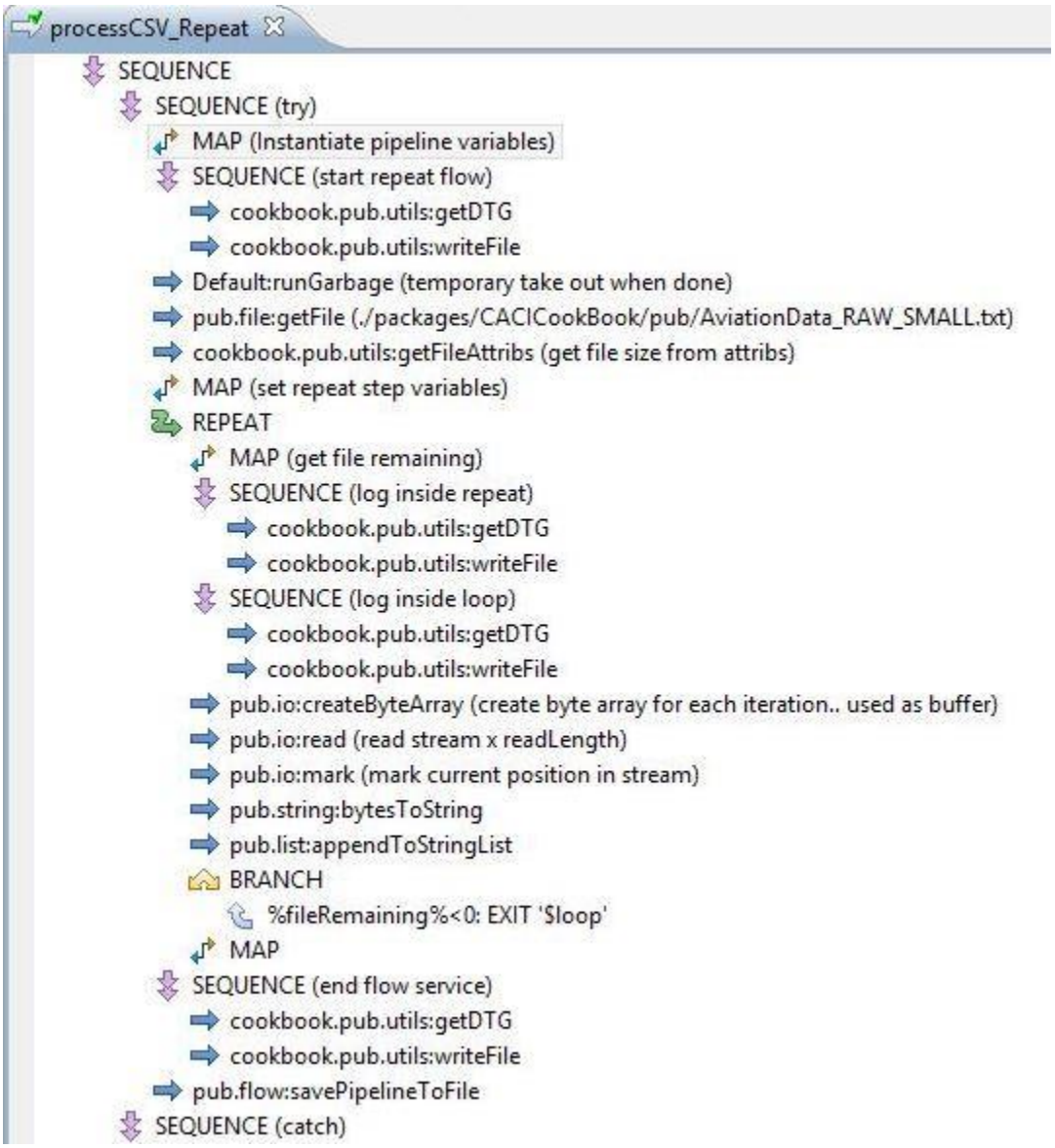
The Repeat Step, set to repeat on success, is used most often to read and process large files. The basic steps involve opening a file stream using utilities in wm.pub.file and then leveraging the utilities in wm.pub.io to parse through the stream by reading a specified amount of data and using markers to mark the current location in the file until the next Repeat Step fires. This type of Repeat Step is most often used when it is desirable to continue activity.

Tips and Tricks – Repeat Step

- Replicates Java Do/While statement
- Design an exit strategy for the Repeat step. For parsing files, determine remaining file contents left to parse. When file contents length is equal or less than zero (=<0), use a branch and an exit step.
- Failure to include an exit strategy for the Repeat step will result in an endless Loop type of condition on the server and could potentially result in an out-of-memory error.

The following example uses a Repeat step to parse a large file by extracting one portion of the file at a time from a stream. Typically, the file processing service parses the file, and inserts the data into a database or converts the file into usable chunks for further processing. Flat file delimited files typically maintain the same file structure from top to bottom and the developer must assume that the file is readable, able to be parsed, and well-formed.

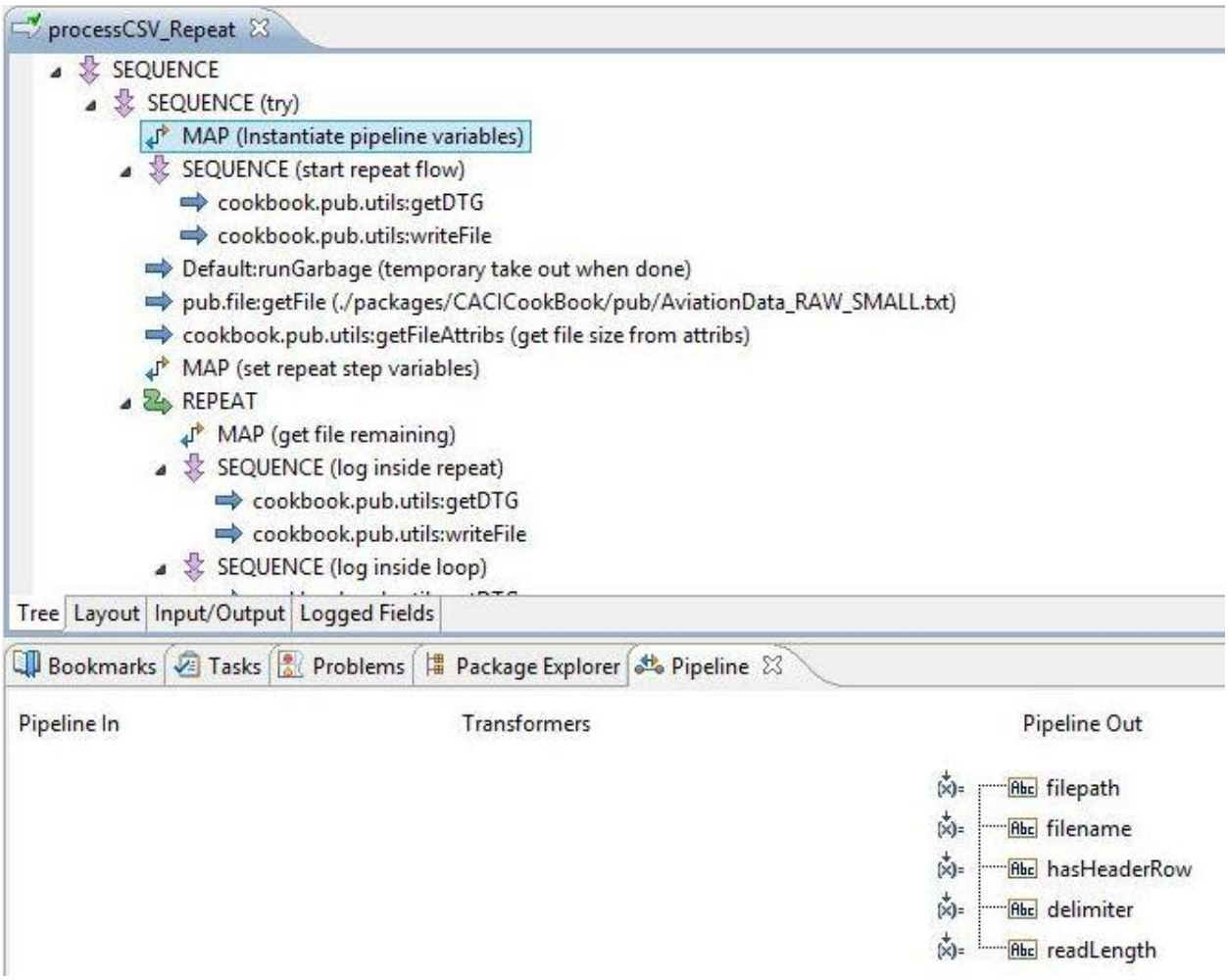
Repeat Step Graphic 1



The Flow, processCSV_Repeat uses the same data set as the Loop Flow illustrated in the previous section. The design difference is that: the Loop iterates over an array of data and the repeat continues to read the file stream until there are no longer any bytes to read. This is not automatic. The Flow initially retrieves the file attribute for file size and subtracts the amount read during each repeat cycle. When the amount is negative, it indicates that no further processing is possible and

exits the Repeat step. Failure to account for this will result in a java.io.Exception. Further file processing is not shown. This Flow demonstrates large file streaming. The first set of graphics document the pre-Repeat steps and the second set document the child steps of the Repeat step.

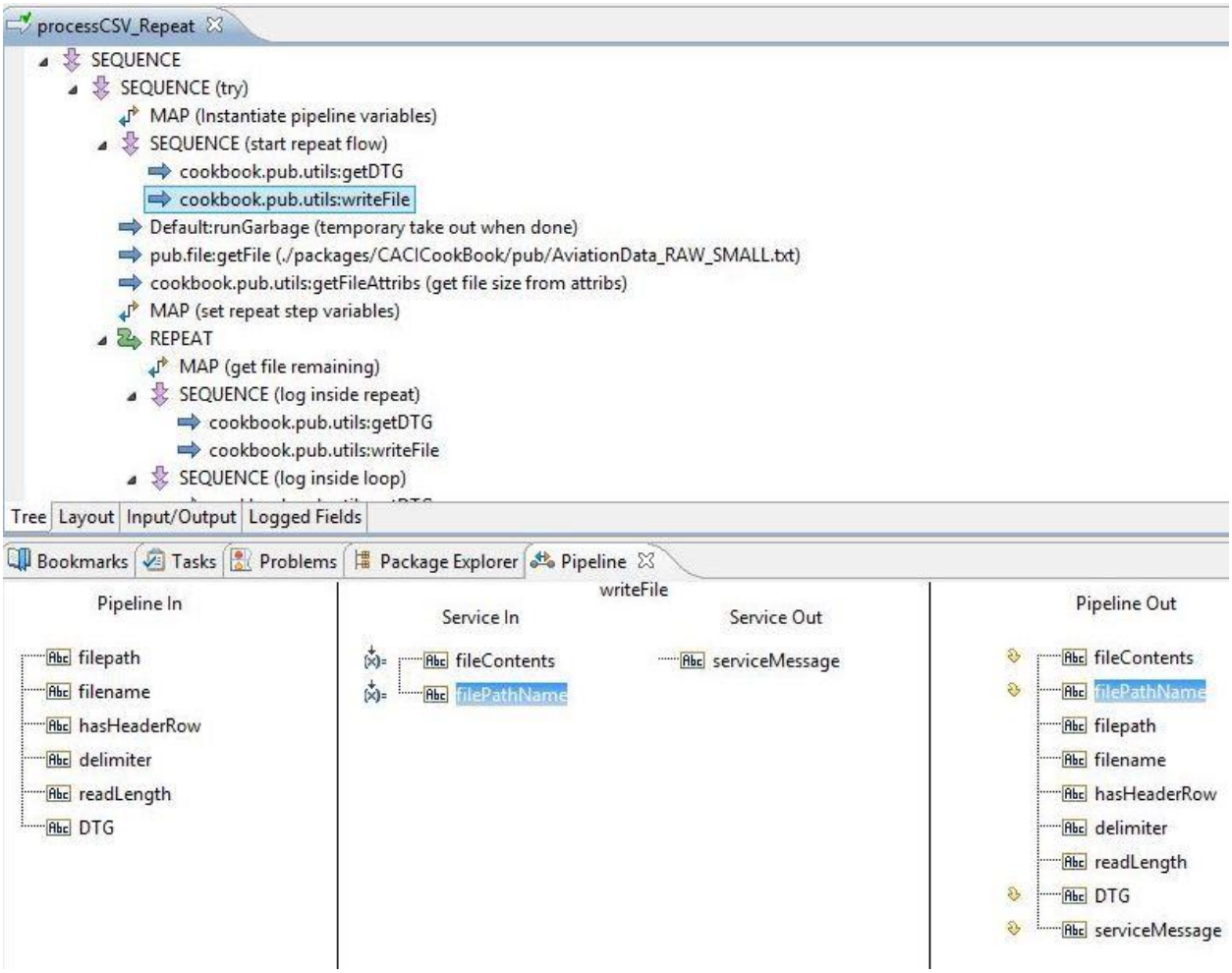
Repeat Step Graphic 2



This Map step instantiates the overall Flow service variables.

Variable	Value
filepath	/packages/pub/CACICookBook/pub/
filename	AviationData_RAW_SMALL.txt
hasHeaderRow	Y
delimiter	|
readLength	10000 (set to whatever makes sense)

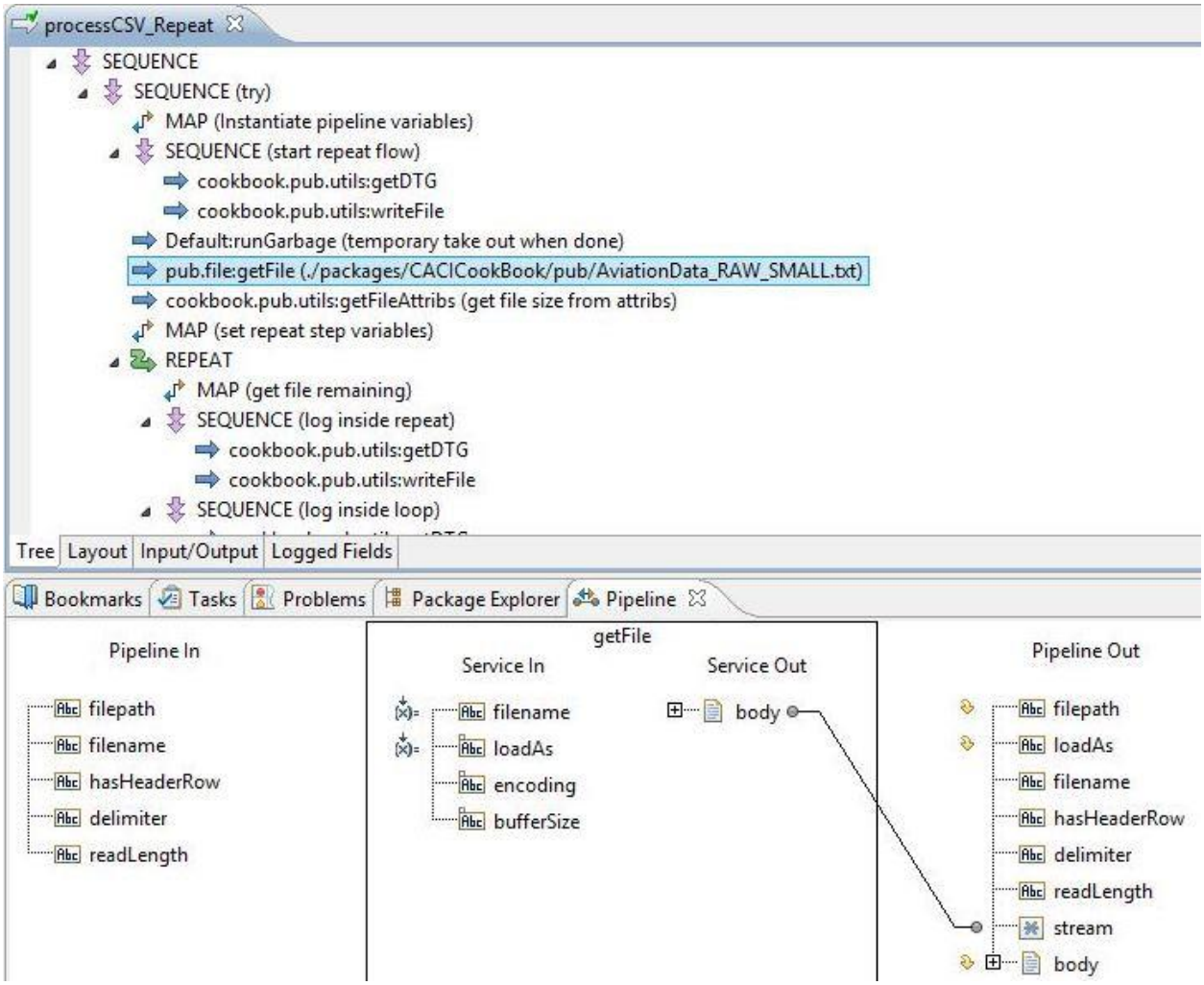
Repeat Step Graphic 3



This second Flow is used several times. It is used to write to a custom log. The writeFile service will write string contents to a file and will append additional string contents each time. The log uses the date time stamp (yyyy-MM-dd HH-mm-ss.SSS) %DTG%.

Variable	Value
fileContents	%DTG% Starting Flow service processCSV_Repeat
filePathName	/packages/pub/processCSV_Repeat.log

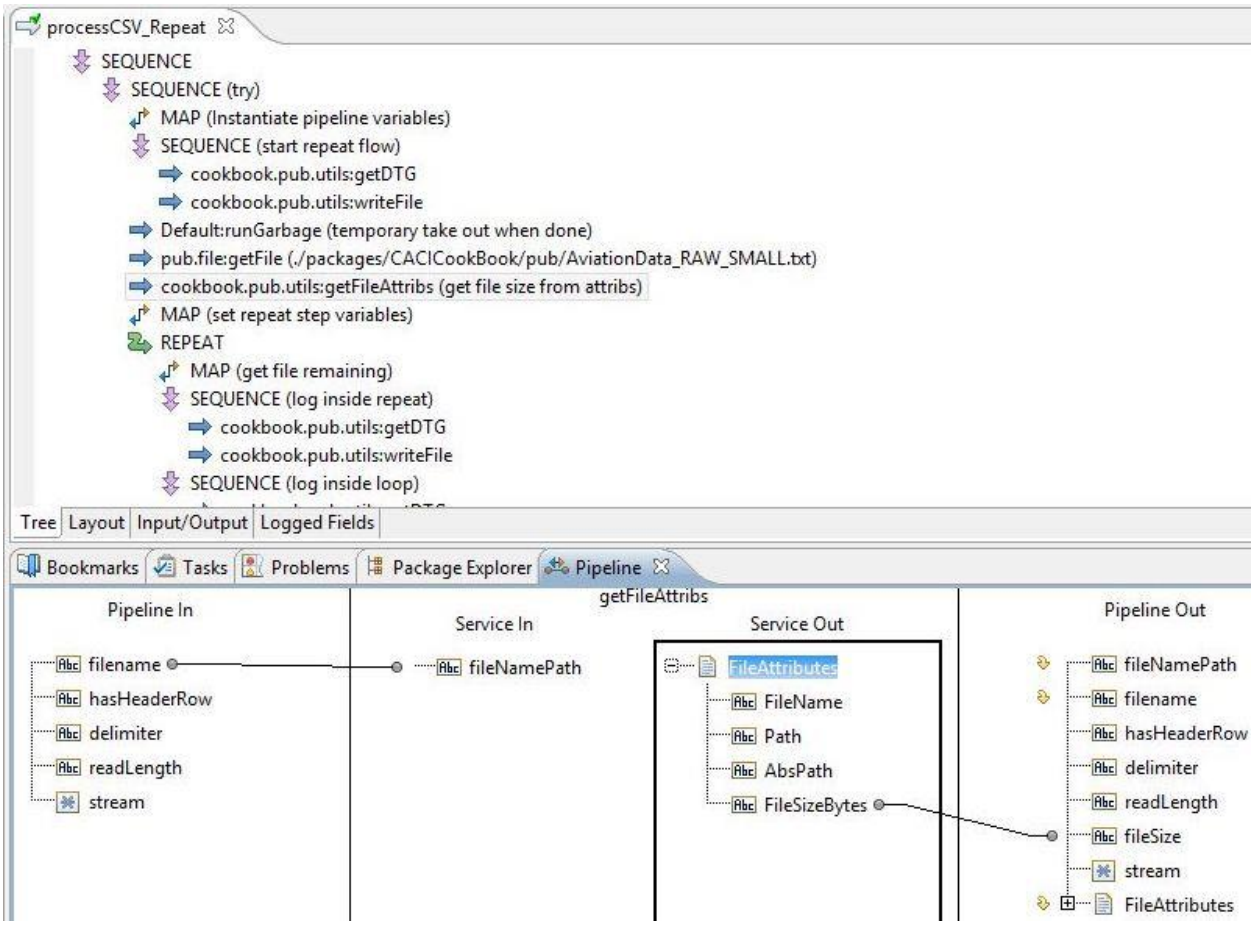
Repeat Step Graphic 4



This step uses the built-in file service, pub.Flow:getFile. It is set to load the file as a stream.

Variable	Value
filename	/packages/pub/CACICookBook/pub/AviationData_RAW_SMALL.txt
loadAs	stream (choices are bytes, stream or string)

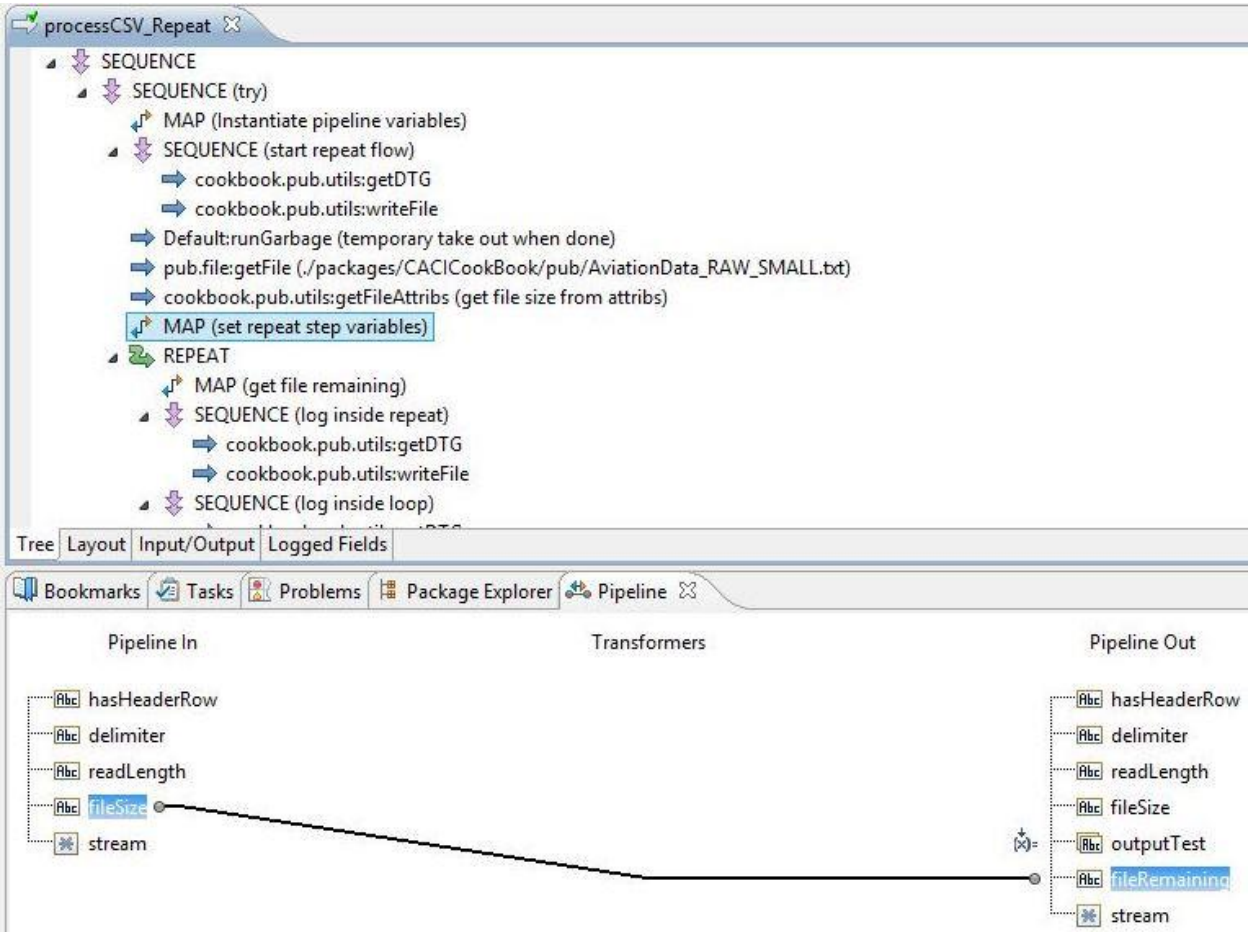
Repeat Step Graphic 5



This Java service step provides the file attributes for a given file. This is needed to determine how many bytes are available to read. Attempting to read bytes past those available will result in a server error.

Variable	Value
filePathName	/packages/pub/CACICookBook/pub/AviationData_RAW_SMALL.txt

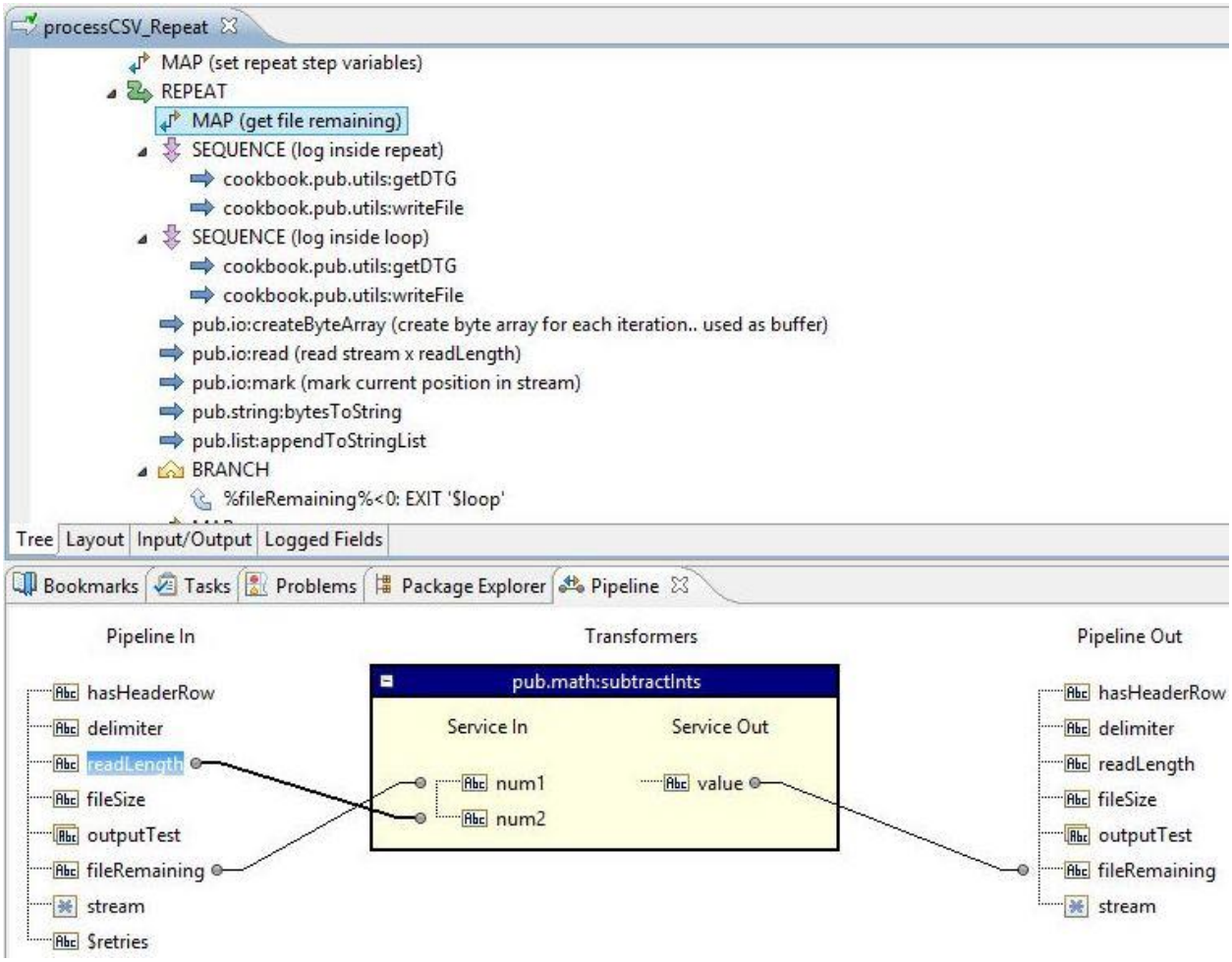
Repeat Step Graphic 6



This step instantiates a String array, outputTest. It will be used to append records during each successful Repeat step.

“filesize” is mapped to fileRemaining for processing.

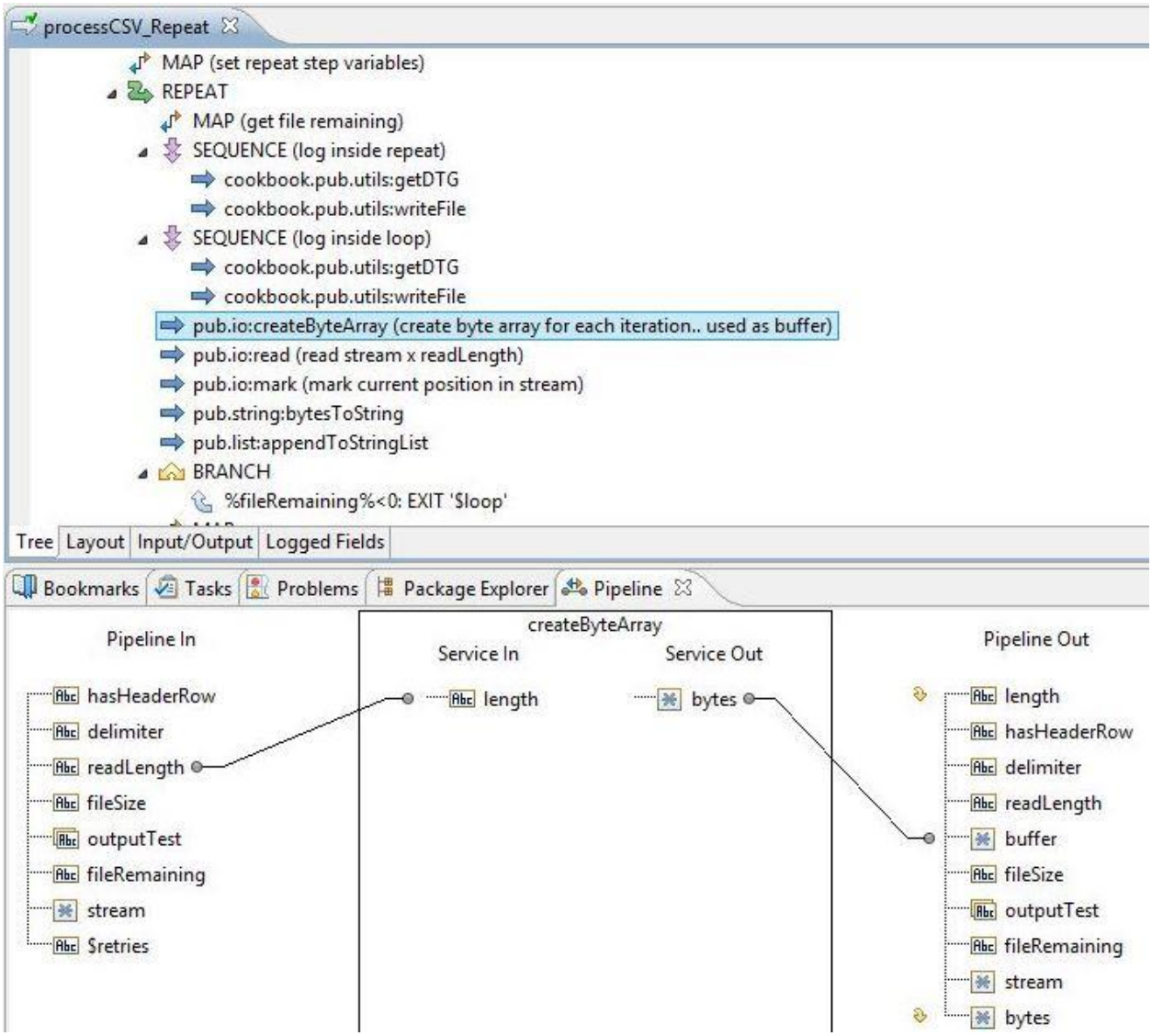
Repeat Child Step Graphic 1



This is the first child step in the Repeat step. It will subtract the number of bytes read from fileRemaining. "fileRemaining" starts out with the total number of bytes available. With each iteration, the number of bytes read are subtracted. When fileRemaining becomes negative, there are no more bytes to obtain.

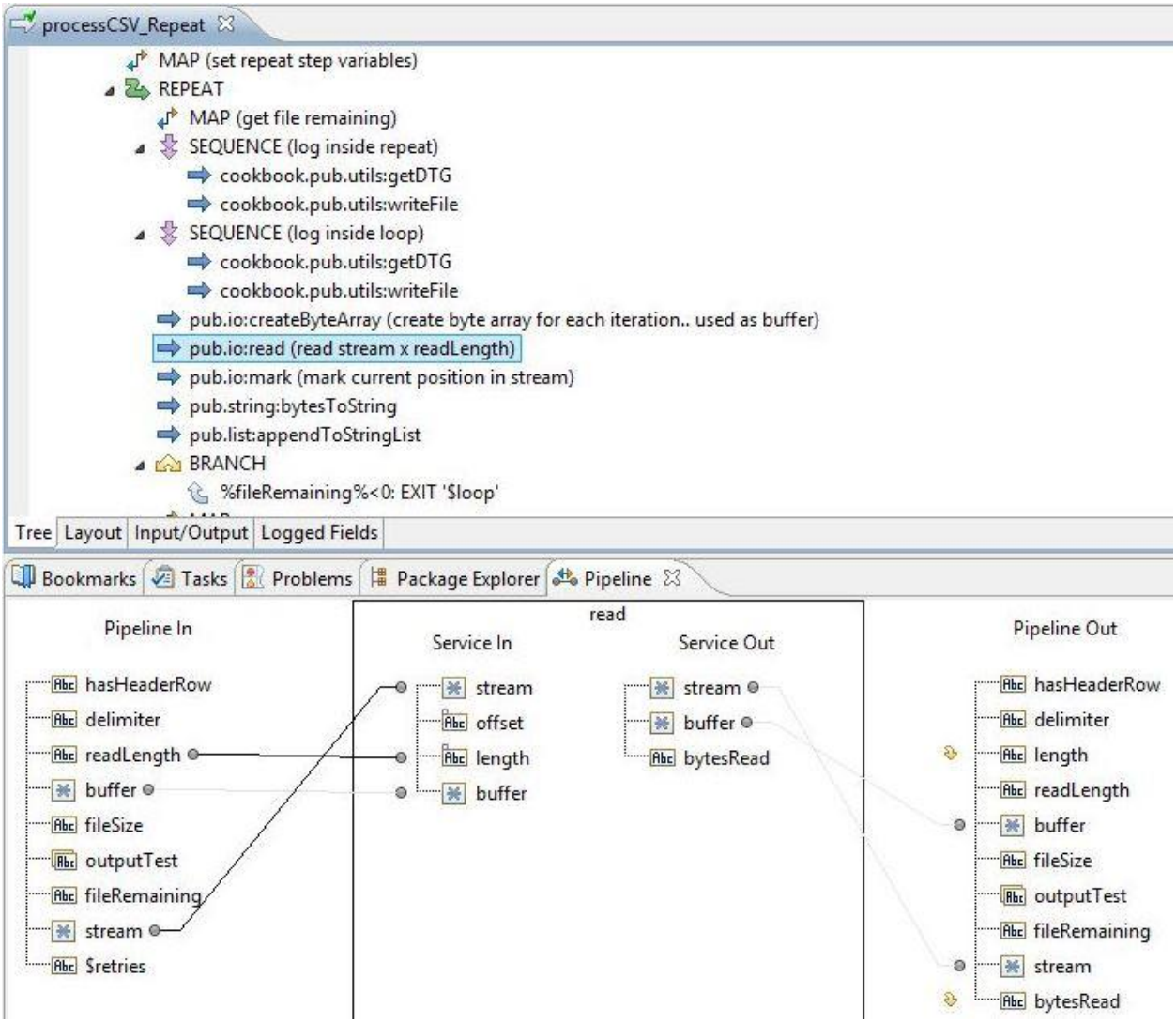
The next sequence is skipped because it uses the log pattern documented earlier.

Repeat Child Step Graphic 2



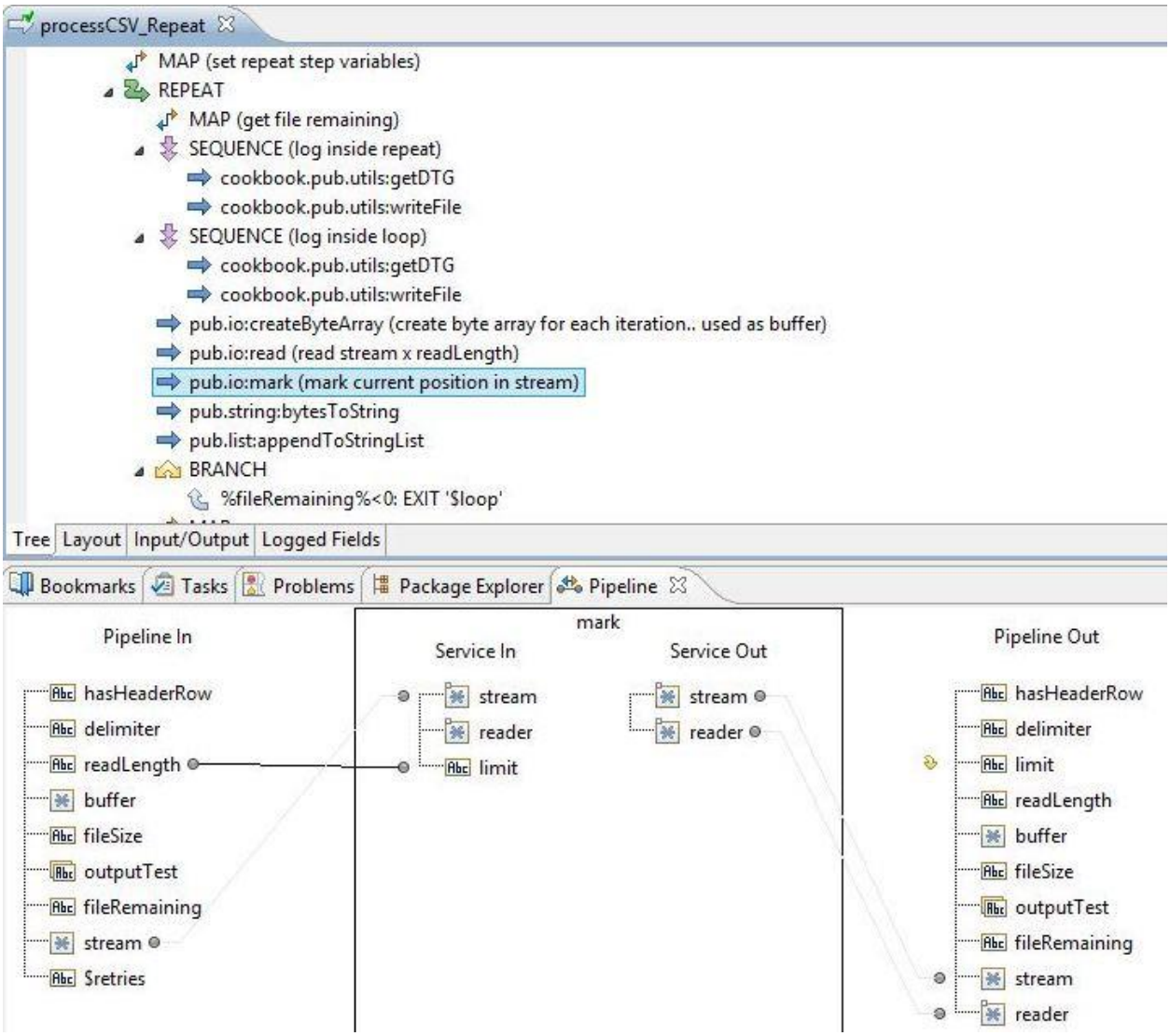
This step creates a byte array needed as an input later. The byte array is named “buffer” to match the upcoming input. The length of the array is exactly the length required.

Repeat Child Step Graphic 3



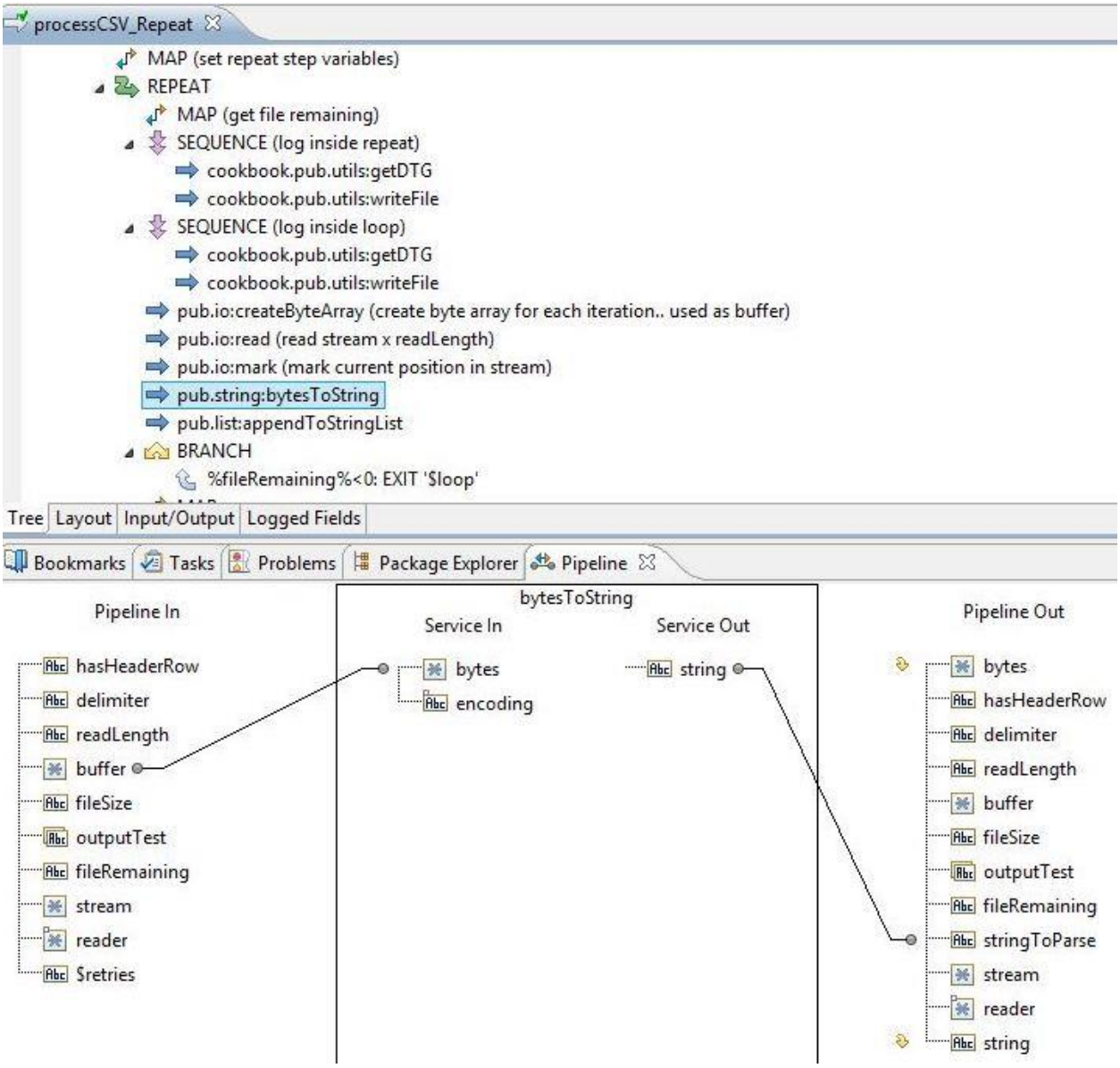
This is the main Flow. It reads in the stream, the amount to read (length) and the buffer object. It outputs the same stream and buffer and indicates the number of bytes read (bytesRead). The buffer is a byte array that contains the number of bytes read during the pub.io:read invocation (based on the number input into length).

Repeat Child Step Graphic 4



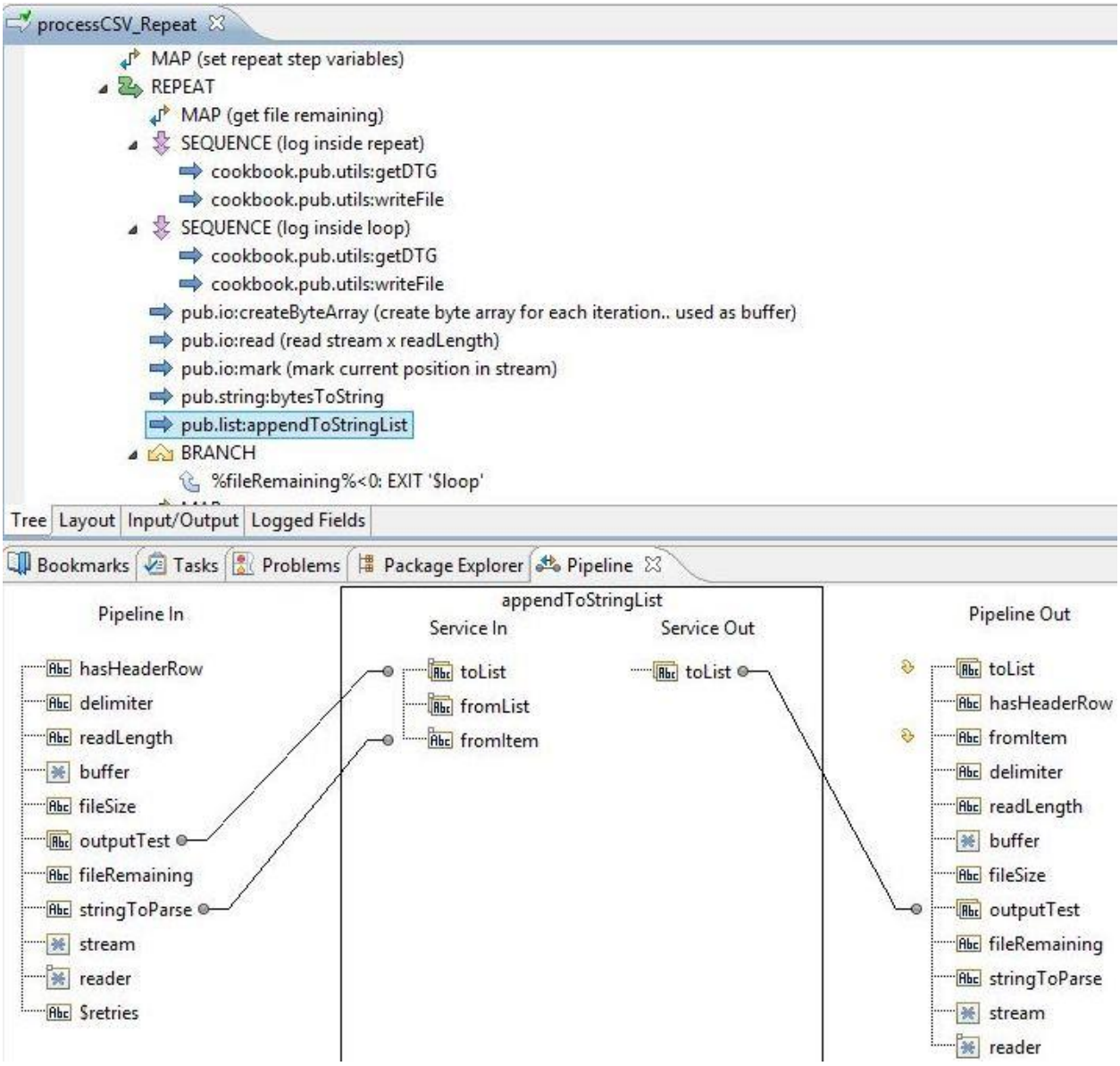
This Flow, pub.io:mark, literally marks the location in the stream so that during the next read operation, the service reads new data. Failure to use this Flow will result in the same data read over and over again. Passing in the *readLength* moves the marker the correct number of bytes. Note, there are other services that can be used to reset the marker. Consult the Integration Server built-in services reference.

Repeat Child Step Graphic 5



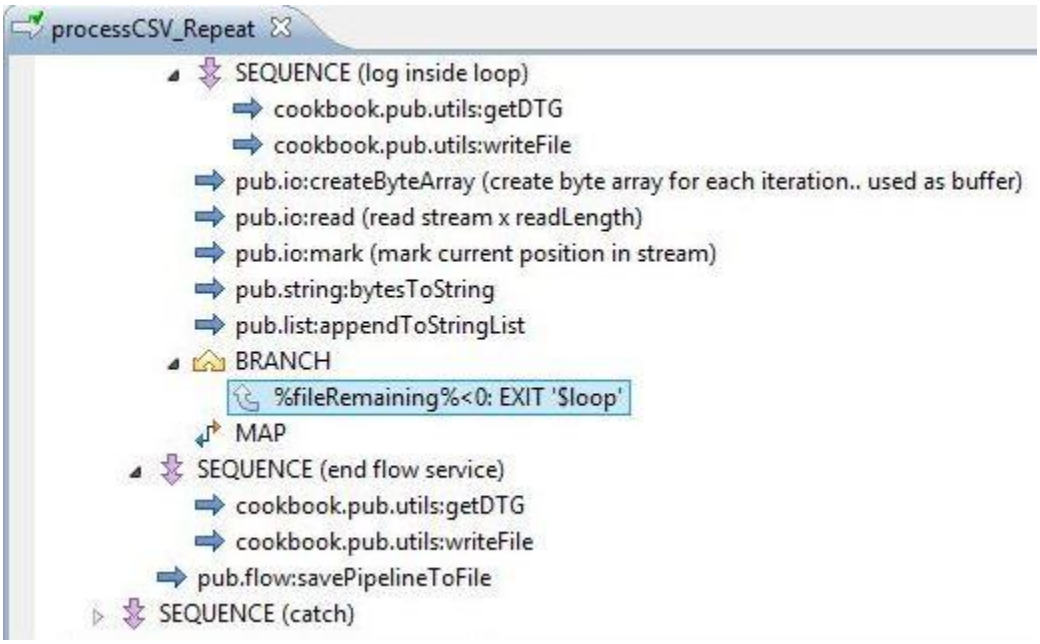
This step passes in variable *buffer*, to the pub.string:bytesToString service. This converts the byte array to a string so the data may be processed. NOTE: No processing logic is included in this Flow.

Repeat Child Step Graphic 6



Each time the Repeat step fires off, the *stringToParse* is appended to string array variable, *outputTest*. This step serves as the processing step. Real life scenarios would take the information, validate and insert into a data warehouse or provide it to other systems for processing. Propagating data between disparate systems remains one of the largest tasks by ESBs. A system or record contains the master record and the ESB updates corresponding systems with either a pointer to the original record or a copy, depending on security constraints and business requirements. A typical use case is an HR system that maintains an employee's address list. If an employee moves, the system of record is updated. This information is propogated to the payroll system, retirement plan administrator, health care, etc.

Repeat Child Step Graphic 7



During each Repeat, the Branch step evaluates if fileRemaining is less than zero. Each iteration reduces the total file bytes available by the readLength (10000 in this case). The last iteration will probably be less than 10000 bytes. "fileRemaining" minus 10000 will result in a negative number during the last valid iteration. The branch will trigger the exit Loop step at that point.

The Map step following the Branch drops unneeded pipeline variables. The next sequence logs the end of the iteration.

The savePipelineToFile call saves the pipeline for testing purposes but should be removed.

The following lists or recaps recap the Repeat step fundamentals or requirements to parse files:

1. Path and name of file to parse
2. Total length of target file
3. Determine if a stream or reader object will be used
4. Logic required to parse the section of the file extracted during each iteration of the Repeat Step
    a. Determine how much information will be parsed in each iteration. Since the Flow service uses a stream, this will be determined by length of read (readLength).
    b. Logic to rebuild last/first lines. This is necessary because the parser will process 1 line at a time. Each line is variable in length (not fixed unlike EDI).
    c. Logic to process the data (unique for each business case)
    d. Potential logic to save the file parsing place in case memory overload conditions occur.

8.6 EXIT STEP

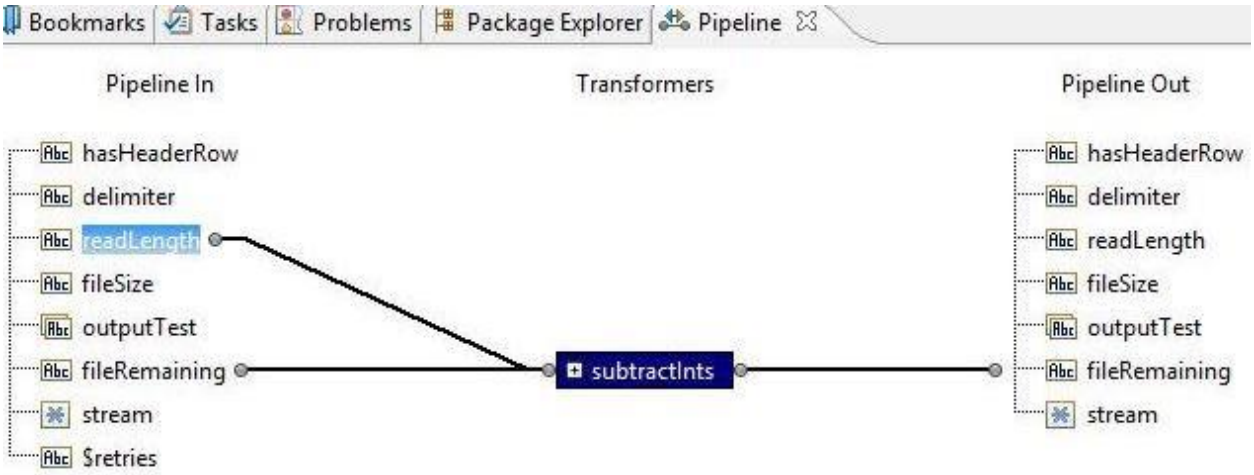
The Exit step is used to exit a Flow, Loop or Repeat step prior to completion. During runtime configuration, the properties can specify success or failure. If the program signals failure, then it will throw an error and if configured, will write a custom error message to the built-in server error log. The developer can choose a custom failure message. The Exit can move the program to the calling parent Flow, from a Loop (or Repeat), or from the immediate Flow (the Flow will revert to the next sequence in the Flow.) The exit function is useful to assist in the structure of the Flow code so that if certain conditions (positive or negative) exist, the code can break at that point and take action as necessary.

The Exit Step is used mainly to exit Loop or Repeat steps. In the previous section, the Repeat step reads from a file stream. It uses the Exit step to exit the Repeat step after logic determines that there is no longer any bytes left to read in the file.

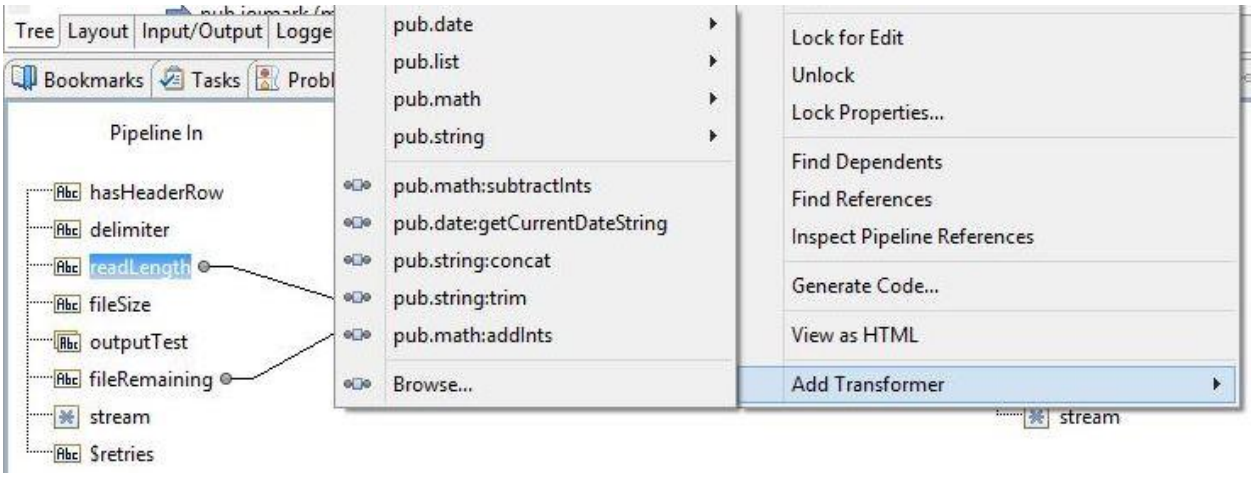
8.7 MAP STEP

Map steps provide a method in which to initialize/instantiate data, view data, map data, or transform data. New pipeline variables can be introduced using this step. Map steps are used to instantiate pipeline variables at the beginning of a Flow service, the first step or a Loop or Repeat step, and normally again during the last step of the aforementioned to drop any unnecessary objects.

Transformers provide a method to manipulate data within a map. The following graphic represents a typical map structure:

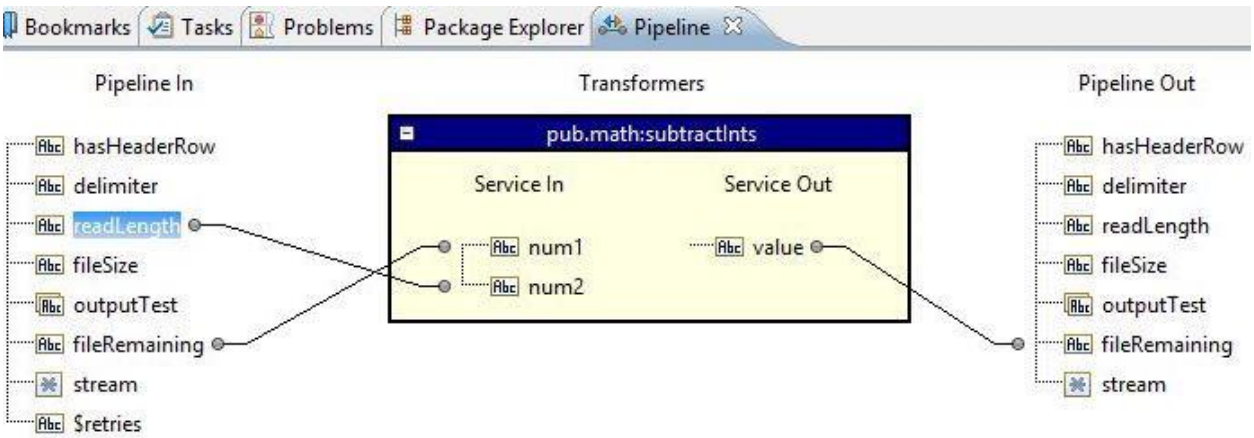


To add a transformer, right mouse click in the white area under "Transformers" and the most common services and the last few that were used will display:



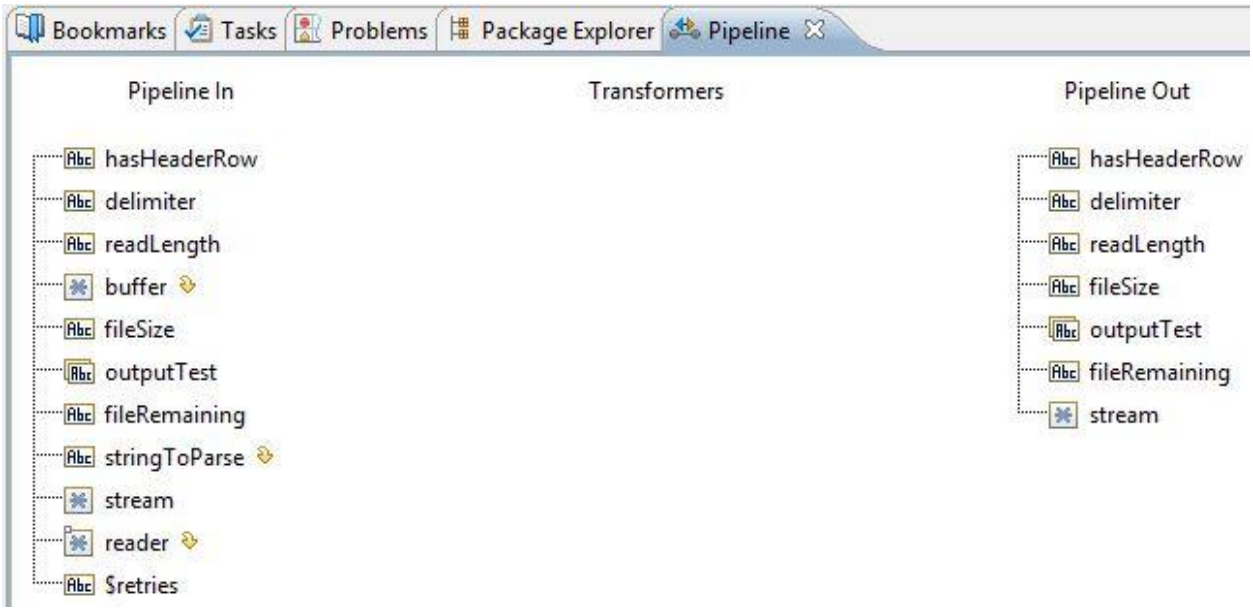
The 4 most common areas of data transformation include date, math, string and list (array) transformations. Note, that any properly formed Flow or Java service can be inserted by using the Browse feature. A properly formed service will have a defined input and output signature.

A transformer when expanded will reveal the inputs and outputs:



Many of the built-in services use names like “inString”, “output”, “value”, etc. While some advocate creating variables that mimic the input/output names, this can cause unforeseen problems since only part of the vector that makes up the pipeline is visible. There are other vector objects that the Server (or IDE) does not provide visibility into. Always use application specific variables (like readLength above).

Finally, Maps are used to drop unneeded pipeline objects. This frees up memory for later garbage collection. Notice that the pipeline objects on the left with the yellow arrows pointing down in the “pipeline in” are not in the “pipeline out” column.



8.8 Java Services: Concepts and Best Practices

The Integration Server is a Java technology-based XML-centric server. Depending on the development team, Flow services may be the prevalent resource developed. Java shops tend to develop Java services to address development without review or regard to the IS built-in services. Built-in services located in the WmPublic package should always be used. These services are in use by thousands of organizations. They have been tested, retested and validated.

The built-in services mainly consist of Java services or Flow services that reference invocations to built-in Java services. Often, these specific services do not have Java source code available. This occurs because the *.Java file is removed after compilation and only the *.class file remains. This is a common attribute for built-in services.

8.8.1 Appropriate Use of Java Services

- When the built-in services do not exist.
- When built-in services do not scale out.
- Java can more efficiently process data mapping using do/while loops. Large data sets may need to be moved to Java. Depending on the situation, significant performance can be gleamed by moving Flow service loops into Java.

8.8.2 Java Services are Super-classed

Integration Server builds and implements Java Services at the folder level. For each folder, the compiler will super class all Java services together. This structure shares import statements. Buggy code in a single Java class will prohibit the successful compilation of the super-classed code.

Java Services have a shared-source area in code. Any methods or procedures that will be shared among the Java services should be put into this public area.

8.8.3 Input/Output Signatures Should Always be Used

Creating Java Services are easy and input/output signatures do not necessarily need to be created. However, adding Input/Output signatures are a best practice. There may be times when a Java Service does not contain a proper signature of any type. Built-in services that are public typically have defined signatures. Internal services often do not. The Java Service must introspect the pipeline object to identify and extract information. If the pipeline does not contain the required data, an error will occur.

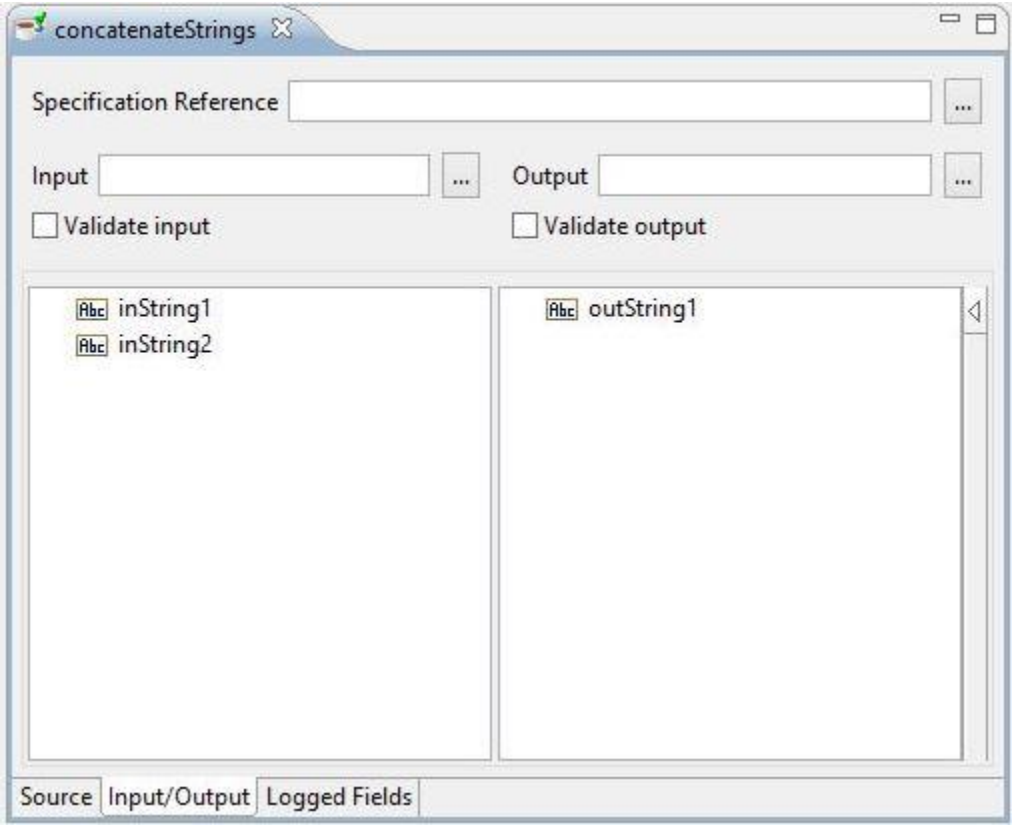
8.8.4 IData Utility

The system uses an IData Object to introduce data into and out of Java Services. It is a little confusing, but the Designer provides an easy way to generate the code required. Consider a simple Java Service that concatenates 2 strings:

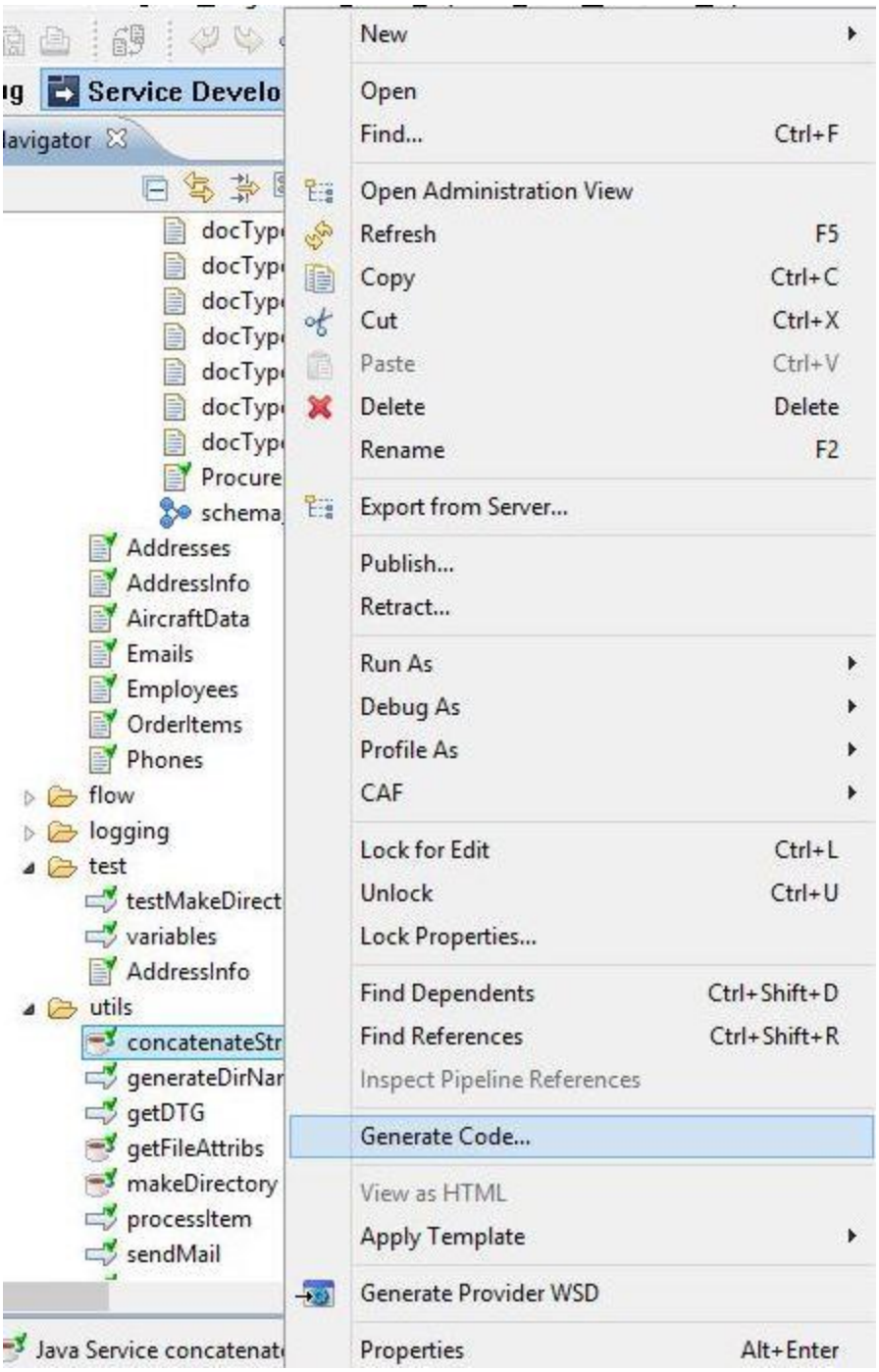
Input: inString1, inString2

Output: outString1

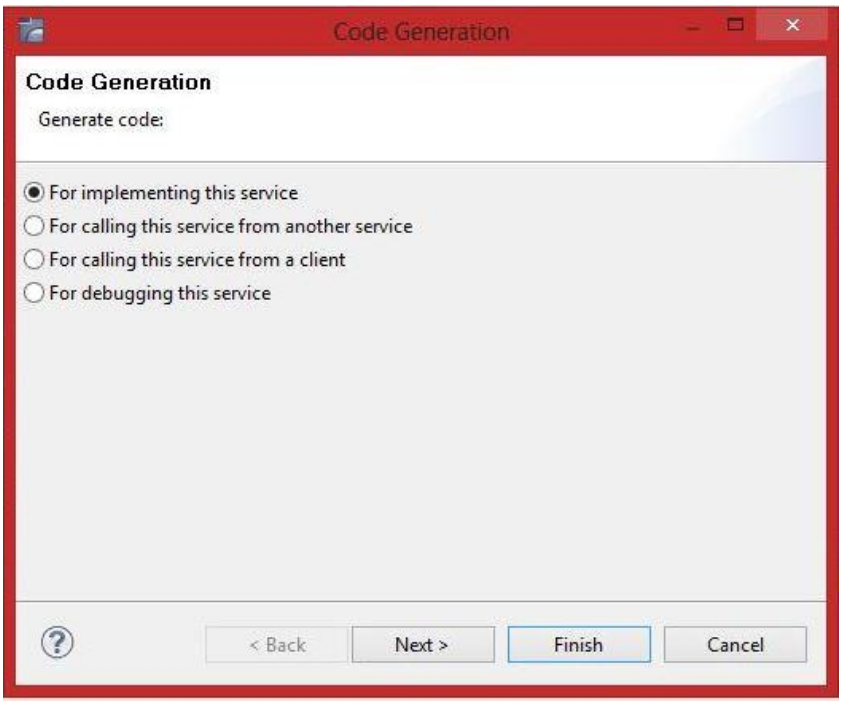
Create a Java Service and name it "concatenateStrings" and set the input/output signature as follows:



.Right mouse click on the Java service you just created and pick the menu item "Generate Code".

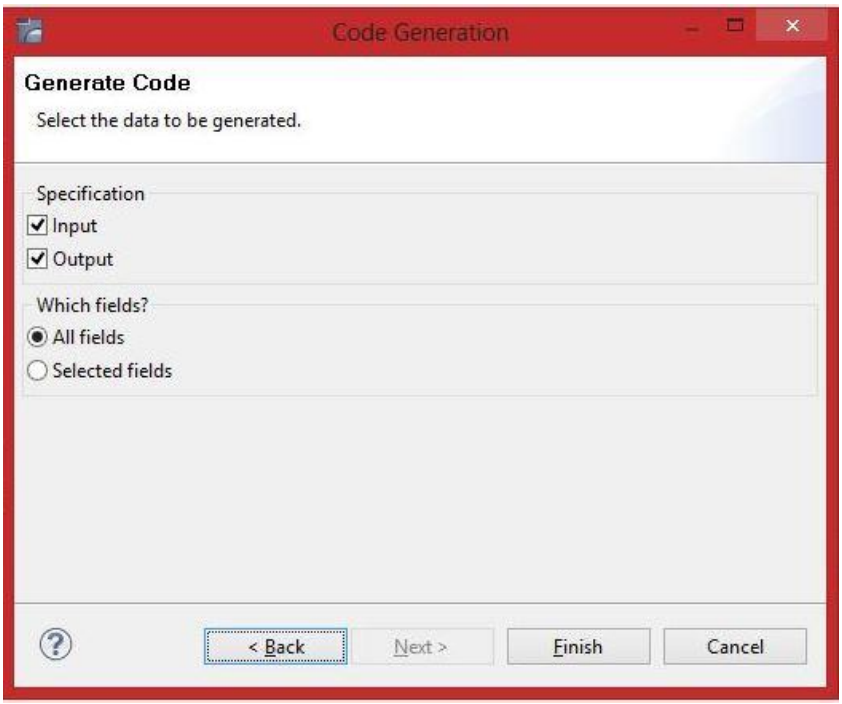


This will activate a wizard. Select the default values for each window.

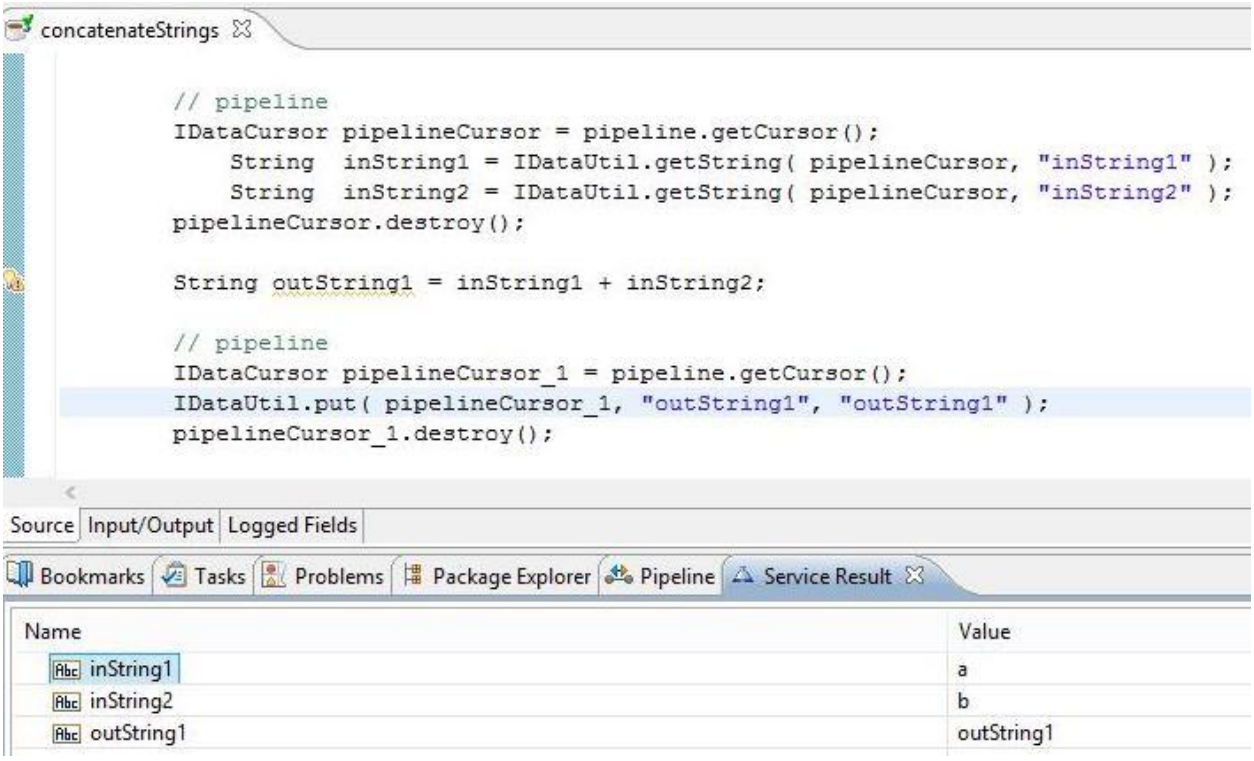


The first option is appropriate. The other options provide specific code for each task. The third choice has additional input to identify what type of client code must be generated.

This will automatically generate the IData Java code required to get data in and out of the Java service. Use the defaults to generate all data.



The code will be placed into the clipboard buffer on your computer. Simply paste the code into the Java window for the "concatenateStrings" service. The window should now resemble:



Running the service with inputs of:

inString1 = "a"
inString2 = "b"

Results in outString1 = "outString1"
The reason for this is because of the following statement:

```
IDataUtil.put( pipelineCursor_1, "outString1", "outString1" );
```

The double quotes around outString1 must be removed. The correct statement:

```
IDataUtil.put( pipelineCursor_1, "outString1", outString1 );
```

Rerunning the Java Service will return the correct outString1 value: This will return the following value:

concatenateStrings

```
// pipeline
IDataCursor pipelineCursor = pipeline.getCursor();
    String  inString1 = IDataUtil.getString( pipelineCursor, "inString1" );
    String  inString2 = IDataUtil.getString( pipelineCursor, "inString2" );
pipelineCursor.destroy();

String outString1 = inString1 + inString2;

// pipeline
IDataCursor pipelineCursor_1 = pipeline.getCursor();
IDataUtil.put( pipelineCursor_1, "outString1", outString1 );
pipelineCursor_1.destroy();
```

Source | Input/Output | Logged Fields

Bookmarks | Tasks | Problems | Package Explorer | Pipeline | Service Result

Name	Value
inString1	a
inString2	b
outString1	ab

The following graphic depicts adding a pipeline value, "outString2" to the pipeline in the Java Service. The "outString2" variable was NOT added to the output signature for the Java service, yet the pipeline contains the variable and value.

concatenateStrings

```
// pipeline
IDataCursor pipelineCursor = pipeline.getCursor();
    String  inString1 = IDataUtil.getString( pipelineCursor, "inString1" );
    String  inString2 = IDataUtil.getString( pipelineCursor, "inString2" );
pipelineCursor.destroy();

String outString1 = inString1 + inString2;

// pipeline
IDataCursor pipelineCursor_1 = pipeline.getCursor();
IDataUtil.put( pipelineCursor_1, "outString1", outString1 );
IDataUtil.put(pipelineCursor_1, "outString2", "Any String of Data");
pipelineCursor_1.destroy();
```

Source | Input/Output | Logged Fields

Bookmarks | Tasks | Problems | Package Explorer | Pipeline | Service Result

Name	Value
inString1	a
inString2	b
outString1	ab
outString2	Any String of Data

This holds true for inputs and outputs. Many of the built-in private Java API calls use the IData cursor to obtain pipeline values. If the variable and value are in the pipeline, then the Java Service can access the variable even if it is not in the signature.

If you choose to not use an input or output signature for your Java service, then your service should contain logic to check for the existence of the required input variables and provide a graceful exit in the event an input is not available.

The biggest reason to remove the source code file and hide inputs/outputs is to keep the source code confidential.

8.8.5 Internal Java Service Debugging

In the event that an internal piece of code does not work, use a Java decompile program to decompile the target source class code. Make careful note of the Java exception. JAD is one program that provides this functionality. JAD will take the class file as input and output a "*.jad" file. It is a text file containing the original Java source code. The Java source code will be a super-class file that will have methods relating to each Java service at the same folder level.

Research of the decompiled file may reveal the error. If not, examine the code to determine what other Java services are called and continue to research, decompile and compare the internal code against the error originally received.

9.0 Recipes For Speed (Prototyping)

9.1 Use Pipeline Objects

The Integration Server provides built-in services to save, restore, or clear pipeline information. The pipeline contains all of the data available to process during service invocation. The following services provide a way to introduce entire data sets into a pipeline service. This proves useful for air-gapped systems in a development environment that do not have access to the data source that may reside in an ERP, SAP, EDW, or other type of system that is unavailable. Data extracts can be introduced into the IS and saved for later use.

The pub.flow folder in the WmPublic package contains several key services. The following table contains a list of the services and a concise description:

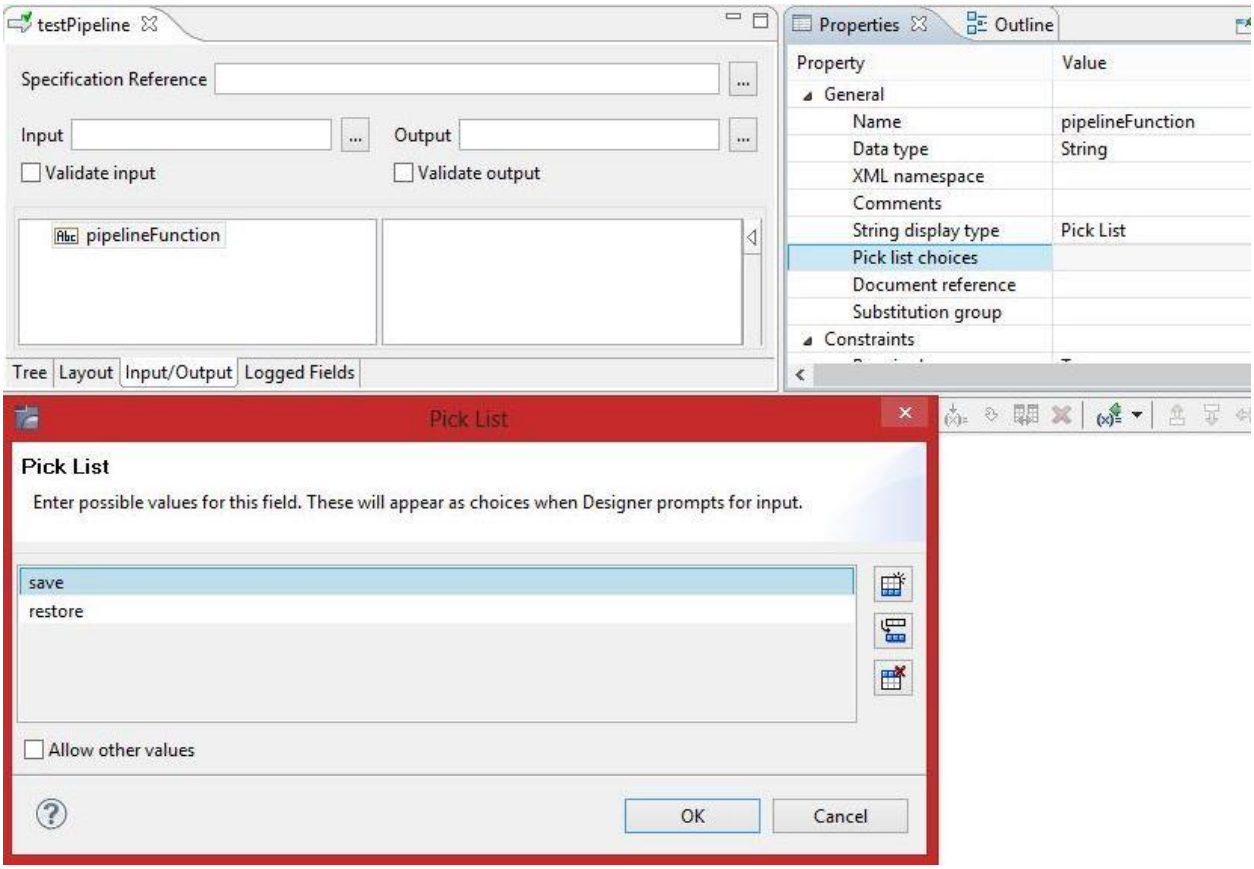
Flow Folder	
pub.flow:clearPipeline	Used to clear the pipeline. Allows preservation of specified pipeline variables. Add variables into string array *preserve.*
pub.flow:restorePipeline	Restores pipeline from memory and aids in quicker testing because the data can be recalled without the need for input.
pub.flow:restorePipelineFromFile	Restores pipeline from a file. Allows developers to rerun services without needing to input data. It can be used to transport the pipeline to other IS servers or share the data with testers.
pub.flow:savePipeline	Saves pipeline to memory and aids in quicker testing because the data can be recalled without the need for input. Information not available after server reboot.
pub.flow:savePipelineToFile	Saves pipeline information into a file and should be used for persistent storage. Information is available after server reboot. Use in conjunction with restorePipelineFromFile.
Comments: **THE MOST USED FOLDER.**	

The Save and Restore functions work in tandem.

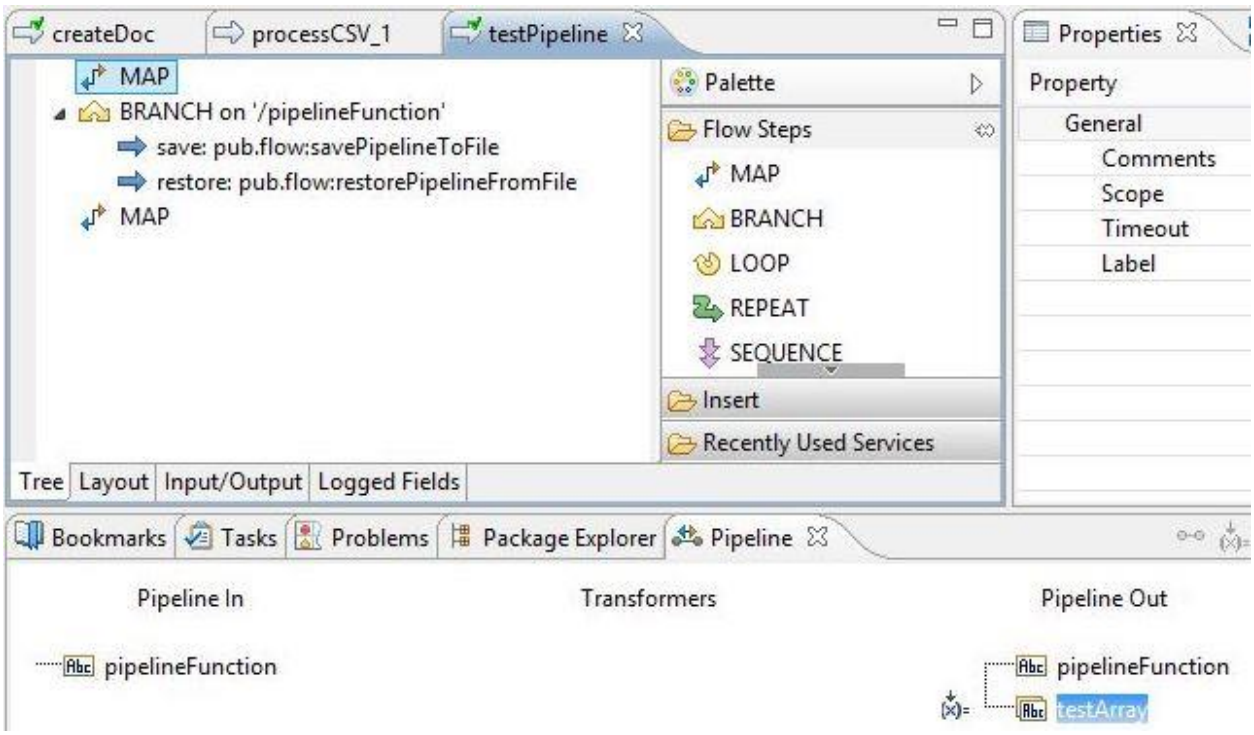
The clear pipeline can be used on any flow service to remove unneeded pipeline information. The developer must specify which objects to retain. All other objects will be removed. This is not a substitute for sound development patterns, such as dropping unnecessary pipeline variables step-by-step. There may be situations where the developer must use other code that may not use stringent discipline, or a parent Flow that calls may child Flow services. Clear pipeline is a best practice for memory management.

The following graphics depict an architecture that supports toggled functionality between save and restore methods. The input, *pipelineFunction*, contains a pick list with the following possible values:

save, restore. This value can be set during the first initial run for the flow. The first Map step in the Flow can set the value for each subsequent run. The expected pattern is to save pipeline and then use restore pipeline to provide input from the retained data.



The following graphic depicts the Pipeline Tree:



9.2 GET DATA MOVING - MAKE XML DOCUMENTS QUICKLY

Integration Server uses "document" objects (IData) to display data in Designer. Creating documents is certainly one of the more error-prone and tedious development tasks. There are several methods to quickly create documents, depending on the type of data.

9.2.1 HOW HUMANS VIEW DATA

Using Developer/Designer, humans see "documents", or other pipeline objects.

List of typical objects include:

Variables String String List String Table Document Document List Document Reference Document Reference List Object Object List	**String**: A single String object. **String List**: An array of String objects **String Table**: Row/Column String Object. Very slow but appropriate in certain circumstances. **Document**: Document object that can contain any of the other objects. Typically instantiated in the pipeline using a map. **Document List**: An array the of same document. Typically instantiated in the pipeline using a map. **Document Reference**: Refers to a predefined document. The document will appear gray in the Designer because of the reference. **Document Reference List**: Refers to a predefined document but creates an array of the referenced document. **Object**: Refers to an object. Common objects include: bytes, stream, and reader. IS does not automatically determine object type. Mapping a byte array to an object and passing that object into a stream object type will cause an error. **Object List**: A list of objects of the same type.

List of objects that may show up in the pipeline include:

Object	Object Type	Comments
	Boolean	Boolean (true/false); cast to String
	Double	64-bit number; cast to String
	Float	32-bit number; cast to String
	Long Integer	64-bit long; cast to String
	Integer	32-bit number; cast to String
	Short	16-bit number; cast to String
	Object	Unknown type.

These objects will need to be type cast using a Java Service.

9.2.2 How Integration Server Views Documents

Integration Server uses XML standards to enforce the integrity of data. A typically XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<AddressInfo>
      <AddressLine1>1000 Main St.</AddressLine1>
      <AddressLine2>Suite 100</AddressLine2>
      <City>Kansas City</City>
      <State>Kansas</State>
      <PostalCode>64105</PostalCode>
</AddressInfo>
```

The Designer View resembles:

The document name translates into the XML ROOT NODE.

9.2.3 Manually Create

Developers can manually create documents by using the Designer tool. There are several SAG documents that provide sufficient instruction (8-2-SP3_Service_Development_Help.pdf).

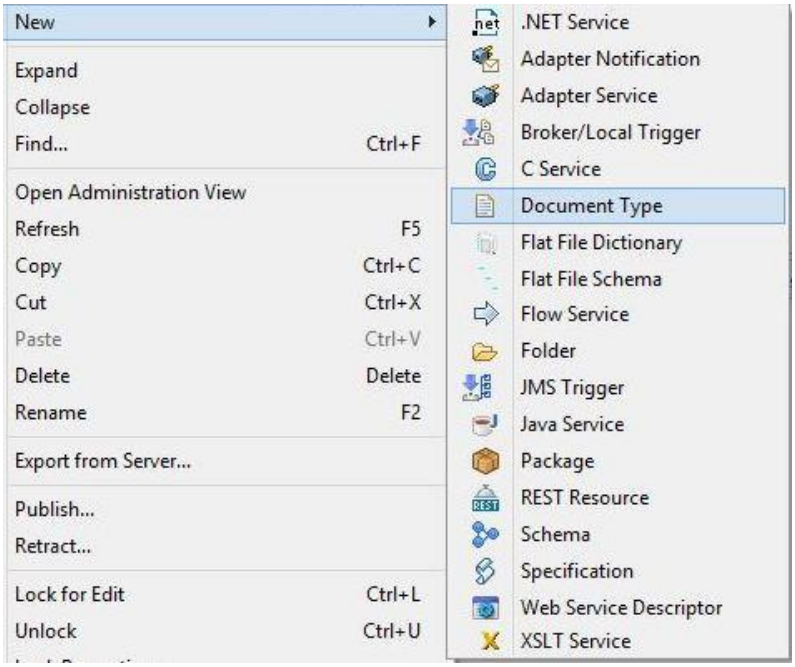
9.2.4 XML Object Import

The information used for this section was obtained from the US Department of Labor, Office of Labor Management Standards (OLMS), http://www.dol.gov/olms/regs/compliance/dsd.htm.

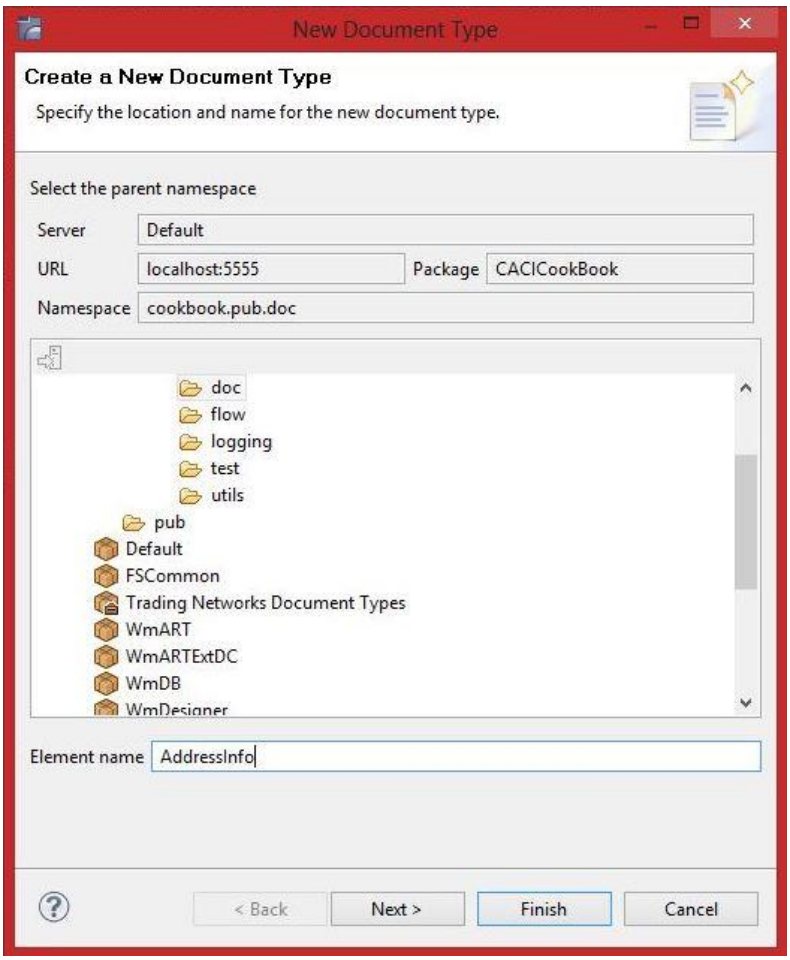
The data import kit was downloaded into the /CACICookBook/pub/DataImportKit/ directory and the code references these files.

XML Document Import

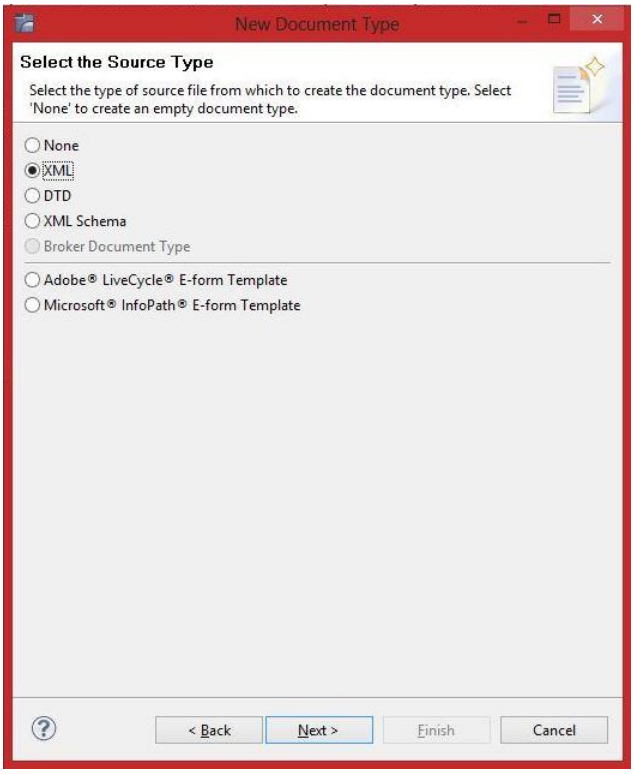
Step 1, right mouse click on the folder to create new document. Choose New >> Document Type from the menu.



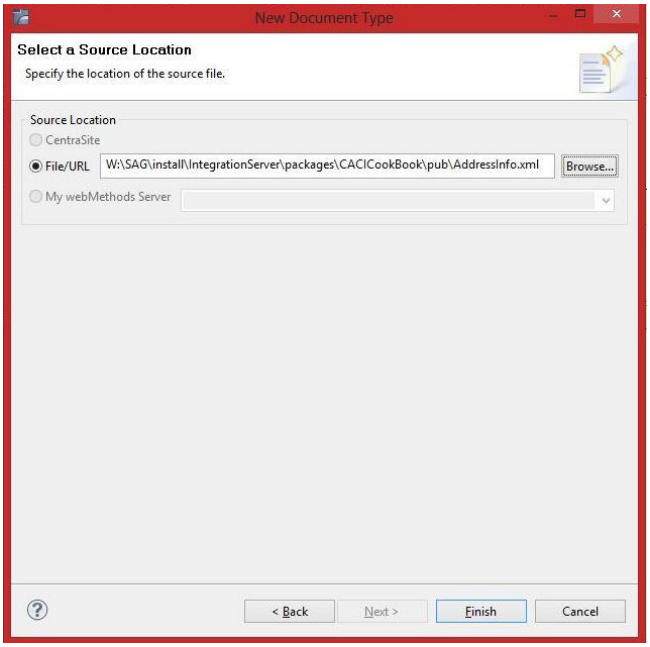
Step 2, type in the name of the new document in the "Element Name" box. Click next.



Step 3, Choose XML. (DTD expects a DTD to be imported, and Schema expects an XSD) and press Next.

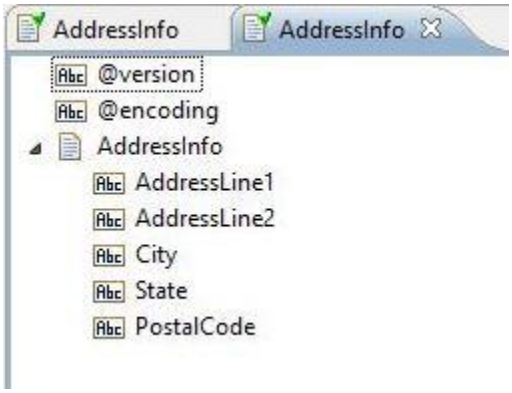


Step 4, Use the Browse button to locate the XML file required. It can also use a URL to download and create. Click on the Finish button.

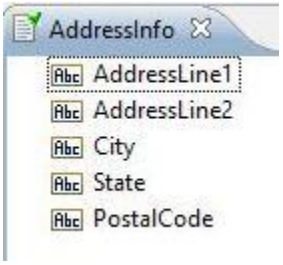


The following warning box may pop up. Click Ok.

The system created the following document. Note that XML attributes were picked up. Attributes have the @ sign in front of the element name. These particular attributes correspond to the attributes in the XML declaration tag: `<?Xml version="1.0" encoding="UTF-8"?>`



Compare this to the document created manually:

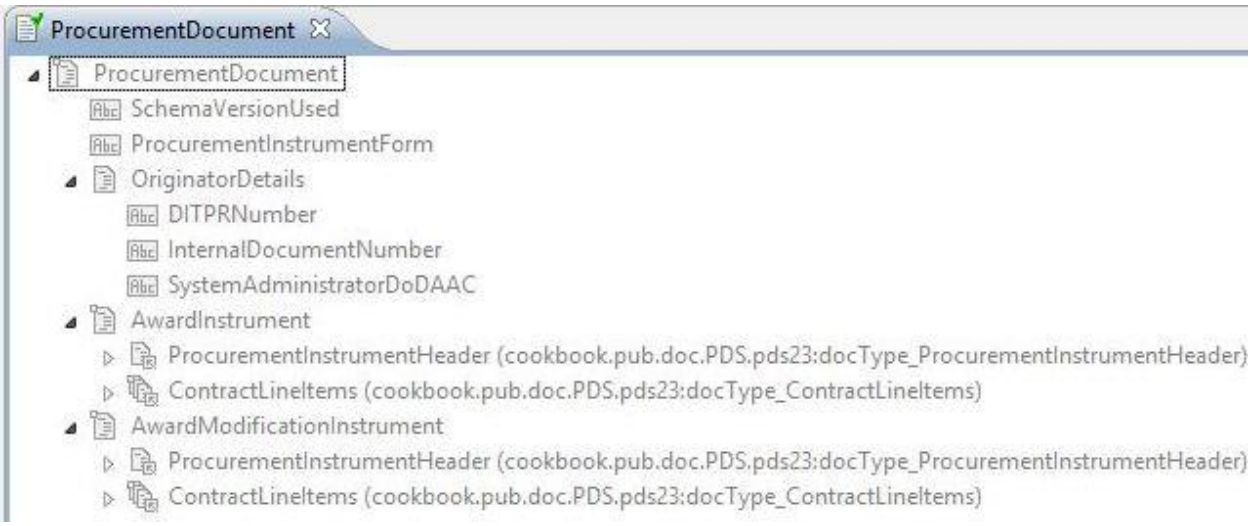


The 2nd manually created document is most likely to appeal to system developers. To remove the additional attribute elements and align the elements, delete the attributes then select all of the child nodes under the AddressInfo node and move them left. Remove the AddressInfo node. If done correctly, the first document will resemble the second document. This is a simple sample. Consider an xml document that contains dozens of elements.

Note that XML files may not contain all of the fields available for a given document. Without the DTD or Schema, or a data guide, there is no way to know for sure if all the elements are present in the data sample.

XML DTD Import

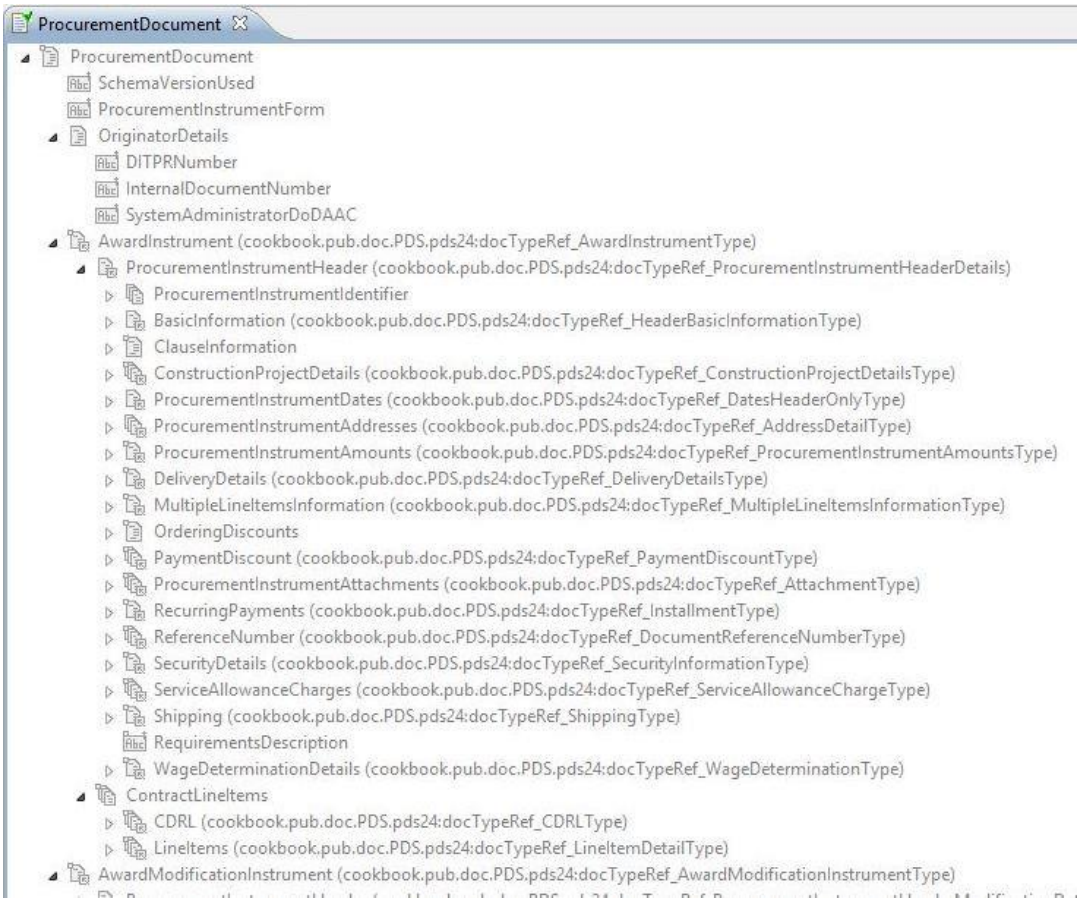
The XML Document Type Definition (DTD) import provides a complete and total structure representation of the XML data. Follow the steps outlined above but use the PDS sample, PDS_Schema_v2.3.dtd, located in the `/CACICookBook/pub/PDS/` folder. Choose DTD instead of XML in the source type window. DTDs do not always create all the document elements. The output for the DTD import should resemble:



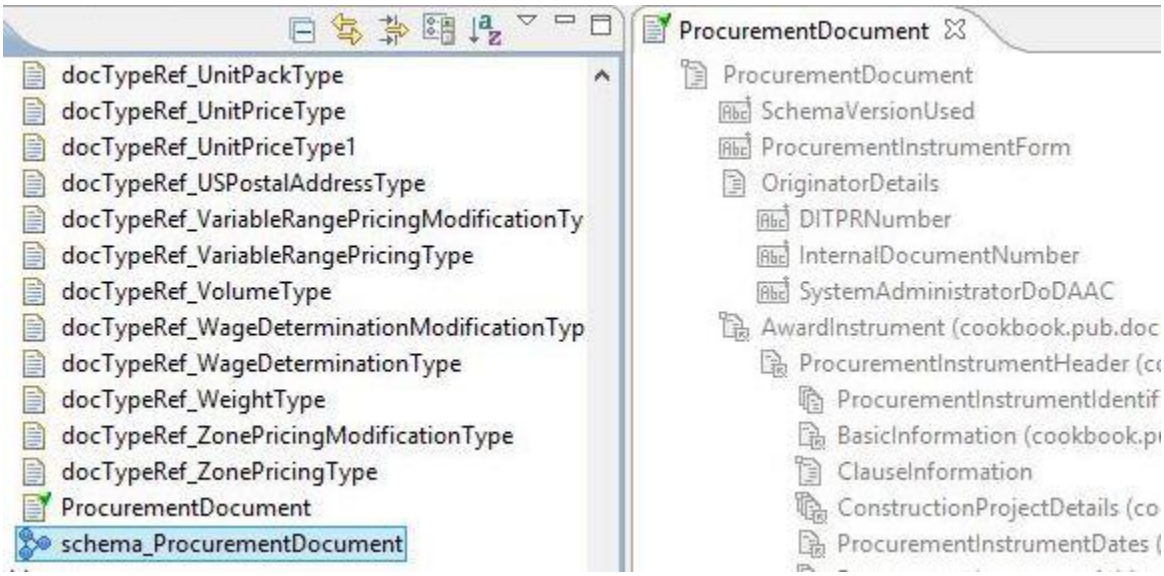
Notice that the root node becomes the top level/parent element.

XML Schema Import

The XML Schema import provides the most complete document object structure in webMethods. In addition to structure, the Schema brings data type constraints which allows for full validation of the submitted XML structure. Follow the steps outlined above but use the PDS sample, PDS_Schema_v2.4.xsd, located in the `/CACICookBook/pub/PDS/` folder. Choose XSD in the source type window. After choosing new document, use the defaults presented in the wizard. The Root node is *ProcurementDocument.* The output for the Schema import should resemble:



Note that the Designer created a much finer-grained XML document. In addition, the system created a Schema object that may be used to validate the XML data in this format.



9.3 Flat File Documents

Comma Separated Values (CSV) represent the bulk of flat file documents. Most flat file documents are referred to as CSV even if commas are not used to delimit the fields. The first row of the file normally will contain the field names, but not always. Each row will contain the field values. If a field is null or does not contain a value, it must still be delimited. Each row is delimited by a carriage return.

The following is a sample of the AddressInfo data structure in flat format, delimited by the pipe character " | ":

```
Address1|Address2|City|State|PostalCode
1000 Main Street|Kansas City|KS|64105
305 Red St|Palo Alta|CA|94556
```

Normally, the architecture demands the use of the flat file adapter to process flat file documents in a production or high capacity environment. This section examines quick prototyping methods.

9.3.1 Create Documents Manually

Just as in the previous section, developers can quickly create document structures to hold the data for a given CSV file. Providing that the structure is valid, a simple tokenizer (set the delimiter to a carriage return) call will separate each row into a string array. Passing the string array into a Loop will expose each row. Pass each row into another tokenizer (set the delimiter to the field delimiter for the given file) to obtain a string array containing the field values. If the file structure is valid, the array will contain a placeholder for each and every field. Fields that are blank or contain null values will still be in the array. See section 8.3 for full details.

9.3.2 Create Documents Automatically

CSV files can contain a high number of fields. This can make manual creation of the document time consuming and potentially can introduce human error. The CACICookBook package contains a Flow service that automatically creates the document structure. There are a few manual steps identified and documented. The methodology to create the document structure includes:

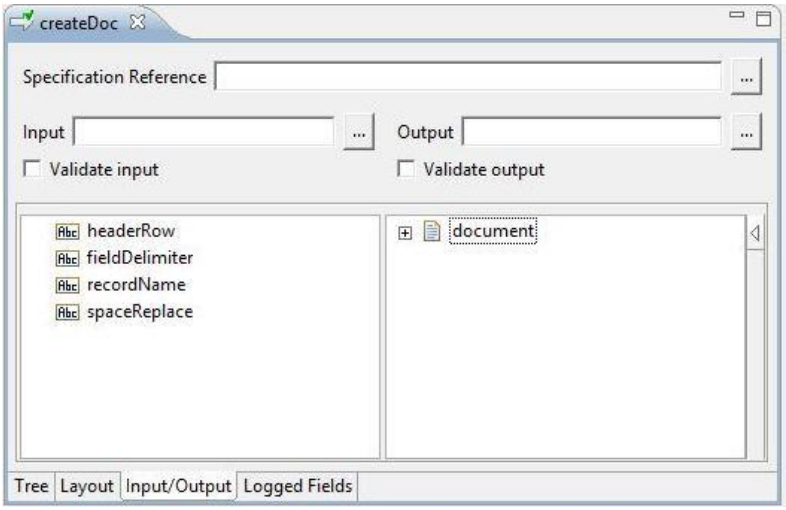
1. Identify header row fields
2. Identify the field delimiter (applies to entire file)
3. Specify a name for the new document
4. Identify space replace. In CSV files, header fields may contain spaces. IS does not support this. So, either specify to remove the spaces or replace the spaces with an underscore.

Navigate to the service, cookbook.pub.adv.createDoc:createDoc. The following represents the input/output signature for the service:

Input Field	Comment
headerRow	Paste the entire header row into this field. Do not add a carriage return.

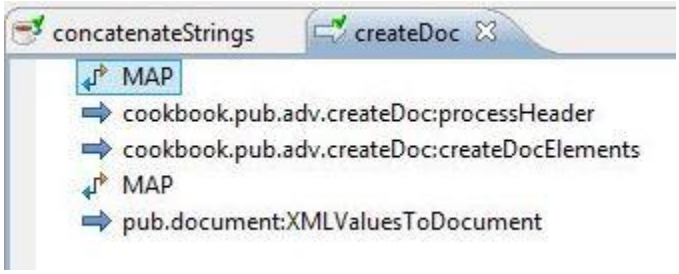
fieldDelimiter	Enter field delimiter. I.e.: a comma " , " or a pipe " | " NO QUOTES
recordName	Name of the document to be created
spaceReplace	Designate whether to remove the spaces or replace with underscores.
Output Field	**Comment**
document	The output is dynamic and dependent on the "recordName" input. The recordName field will become a child node of document.

The Flow service input signature:

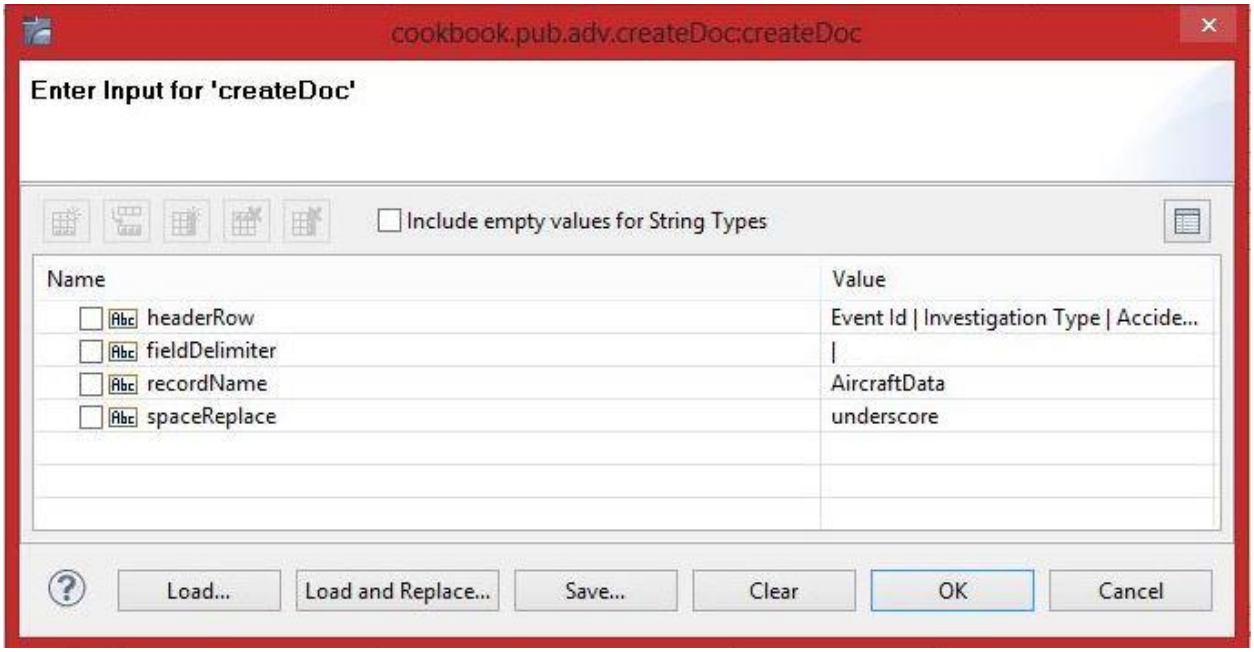


If the process works correctly, the pipeline will contain the new data structure.

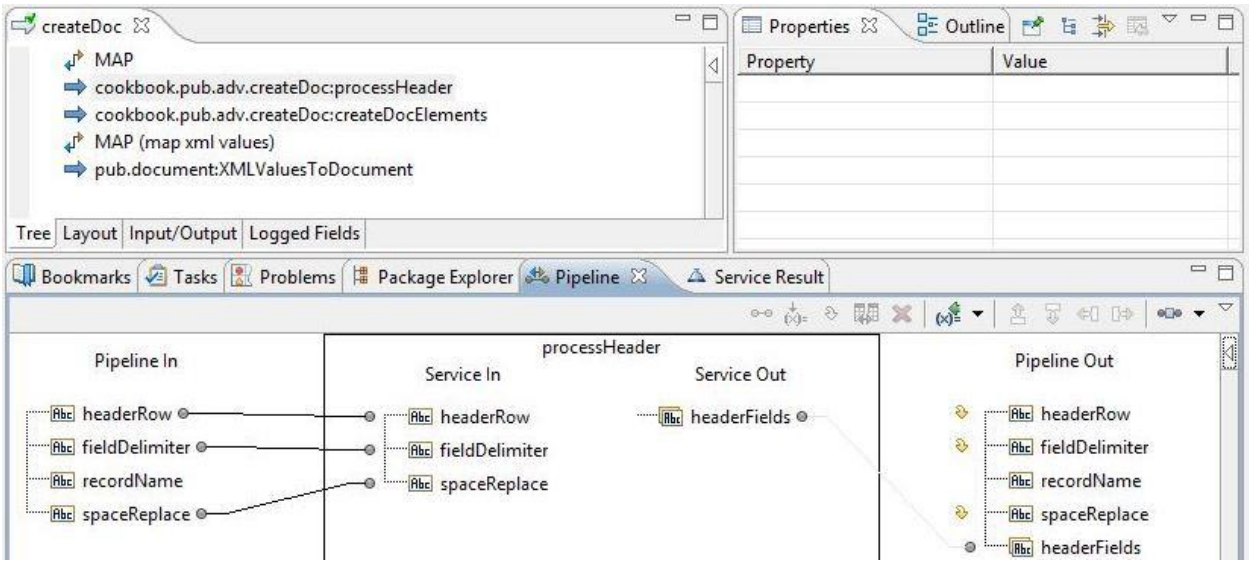
The logic and steps required to produce the new document record includes:



1. Invoke Flow service cookbook.pub.adv.createDoc:createDoc.
    a. Enter in header row for CSV file
    b. Provide delimiter
    c. Provide name for new document
    d. Indicate if spaces should be totally removed or if spaces should be replaced with an underscore.



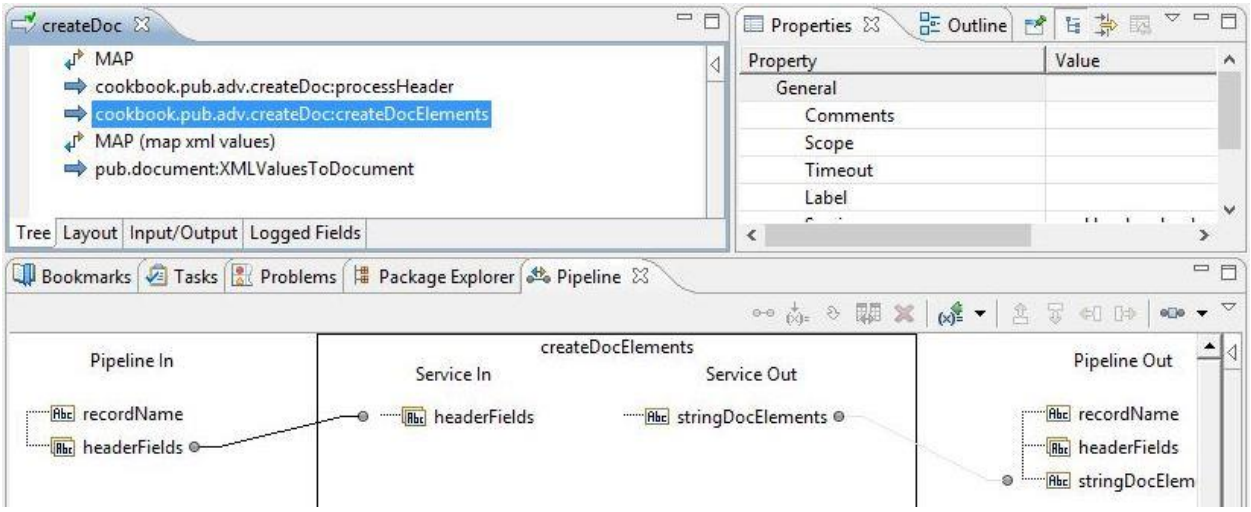
2. The Flow service invokes child Flow cook.pub.adv.createDoc:processHeader. Tokenizes header columns into a string array.



3. The Flow service invokes child Flow cook.pub.adv.createDoc:createDocElements. This Flow creates the core XML elements needed to pass into the pub.document:XMLValuesToDocumentFlow.

Consider the input of headerFields: Address1, Address2, City, State, PostalCode. The string output "stringDocElem" contains the following string:

```
<value name="Address1"></value>
<value name="Address2"></value>
<value name="City"></value>
<value name="State"></value>
<value name="PostalCode"></value>
```



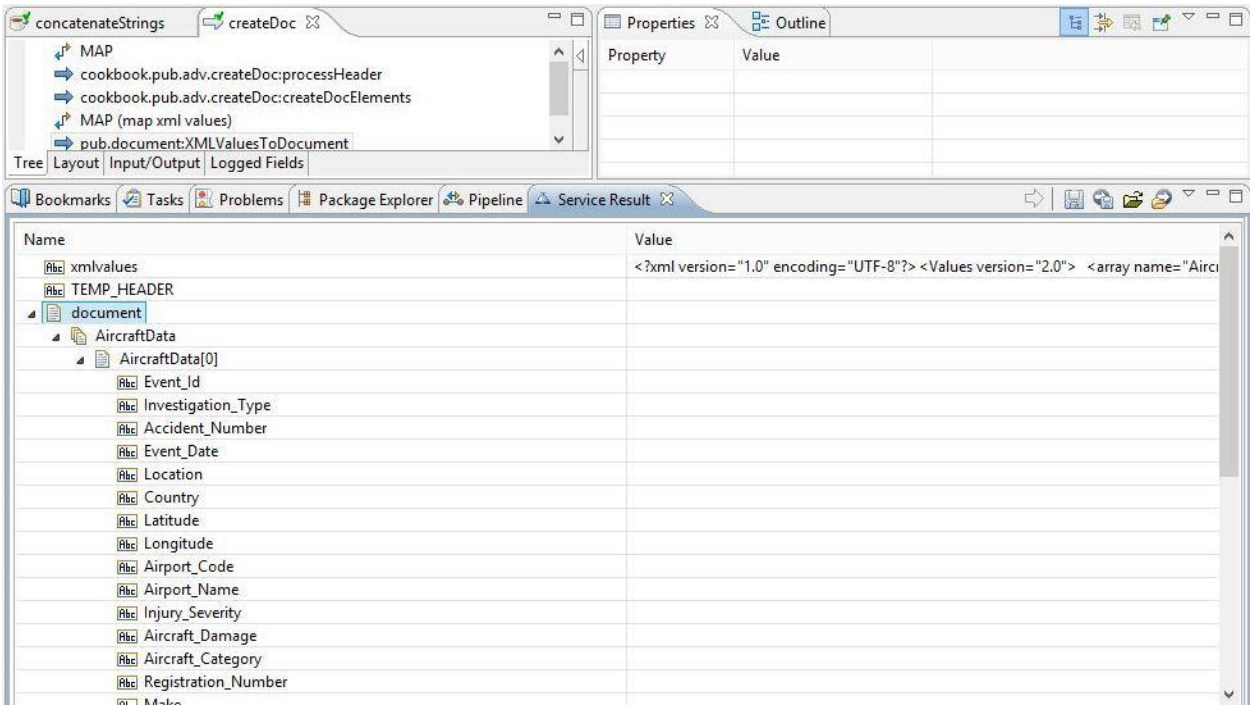
4. The next Flow service step, a map step, sets string element, xmlvalues, to:

```
<?xml version="1.0" encoding="UTF-8"?>
<Values version="2.0">
  <array name="%recordName%" type="record" depth="1">
    <record javaclass="com.wm.util.Values">
 %stringDocElements%
</record>
     </array>
</Values>
```

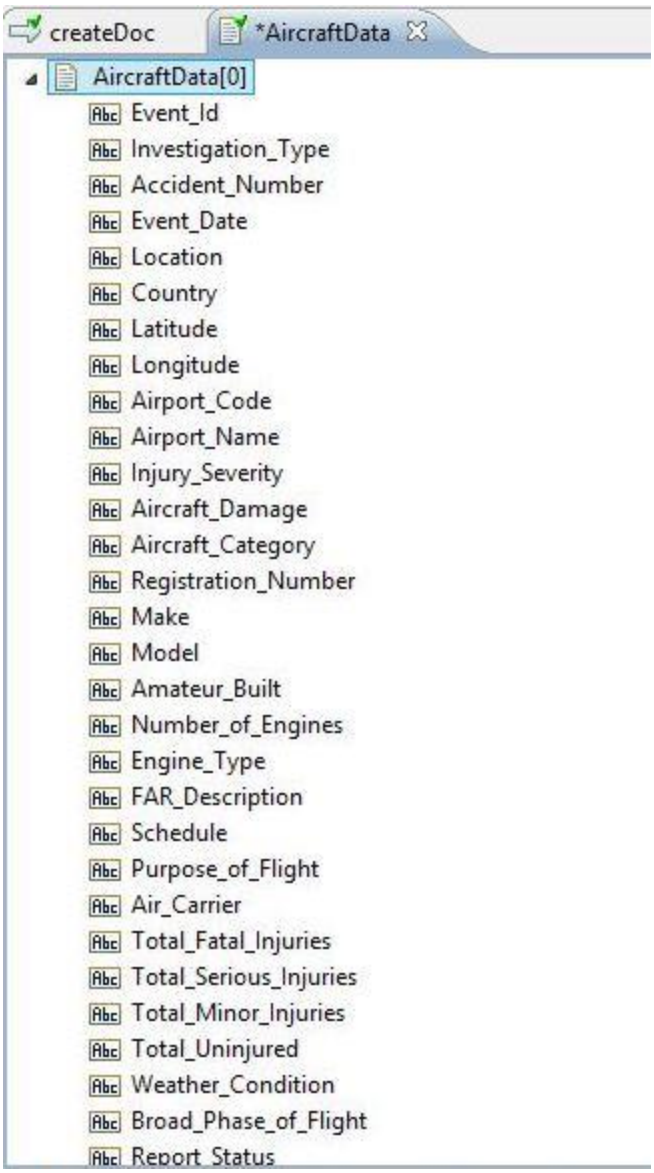
Notice in the variable substitutions for array name and for the body (stringDocElements). These substitutions dynamically name the document child node (under the "document" node output) and it provides the headerFields inputted earlier to create the string elements needed under the "recordName".

5. The last Flow service step passes the string, "xmlvalues" into pub.document:XMLValuesToDocument.
6. At the conclusion of the service invocation, examine the Service Result Tab to locate the document node in the pipeline. Expand the node. The first child node of "document" should be the new document set to "recordName"

Using the prior AircraftData example:



7. Create a new document and give it the same name as field “recordName”. In this case, AircraftData. Select the AircraftData node or the AircraftData[0] node. Paste this into the newly created document.


createDoc
*AircraftData
AircraftData[0]
Event_Id
Investigation_Type
Accident_Number
Event_Date
Location
Country
Latitude
Longitude
Airport_Code
Airport_Name
Injury_Severity
Aircraft_Damage
Aircraft_Category
Registration_Number
Make
Model
Amateur_Built
Number_of_Engines
Engine_Type
FAR_Description
Schedule
Purpose_of_Flight
Air_Carrier
Total_Fatal_Injuries
Total_Serious_Injuries
Total_Minor_Injuries
Total_Uninjured
Weather_Condition
Broad_Phase_of_Flight
Report_Status

8. Select the first to last element (in our example, Event_Id through Publication_Date with the shift key under AircraftData[0] and move them left. Delete the AircraftData[0] element. The record is now clean. It may be referenced as a single document or a document list.

This process can be used to build very large documents with multi-level arrays. Developers can cut and paste column fields from document tables, add delimiters, and use this service to quickly build and prototype webMethods data structures.

9.4 Unit Testing for Results

How many times has a minor change in your code broken everything? A quick and dirty method to testing code starts in the beginning. Effective developers build debugging into their code. The next section addresses which components to use and how to get the most out of different patterns. This section details how to test your code using other means besides Designer. The eclipsed-based

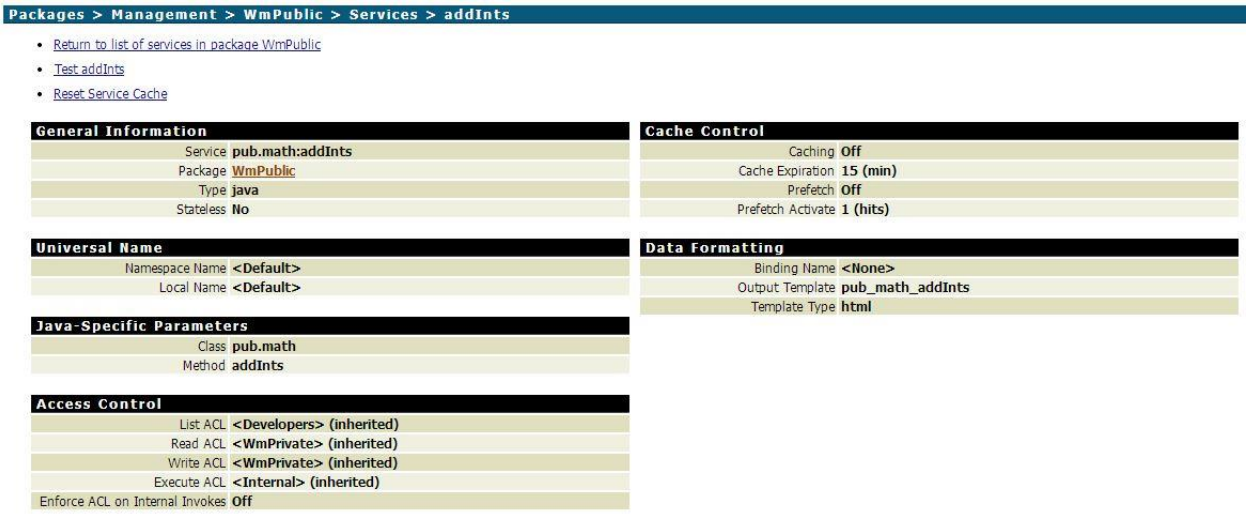
Designer was long-waited by many. The Designer falls short in friendliness and simple functionality compared to the now-deprecated Developer IDE. The Developer excelled in step-by-step debugging and Designer is unable to get close to matching the basic debugging functionality. Much of this is due to the need to adhere to Eclipse functionality. Eclipse changes perspectives, which has proven to be very confusing.

9.4.1 Use Developer for Simple Step-Through Debug

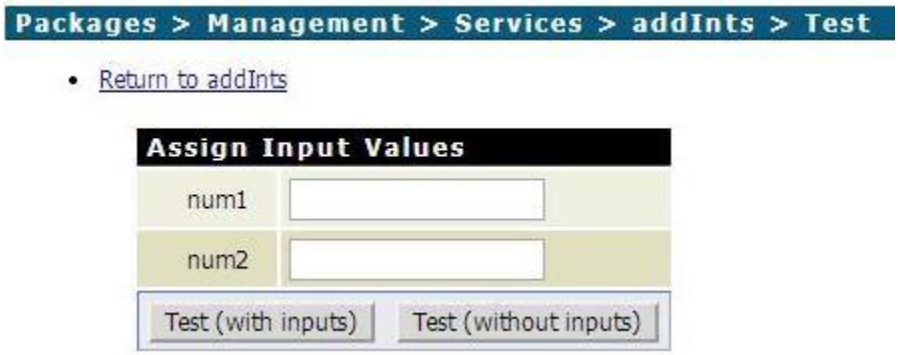
Although deprecated, the Software AG Developer will still work for version 8.2.x. It has not been tested on version 9.x. Stepping through Flows and stepping "INTO" a child Flow is easy to do with Developer. No so with Designer. Frustrated with the lack of visibility into your Flow? Developer even steps through each and every map step to transformer.

9.4.2 Use the Administrative Console

The user who chooses this method will need Admin rights to the webMethods Integration Server console. Navigate to Server >> Server Usage and click on the service of interest. For this test, choose pub.math.addInts. At the top of the page, a clink on the link "test *addInts*".



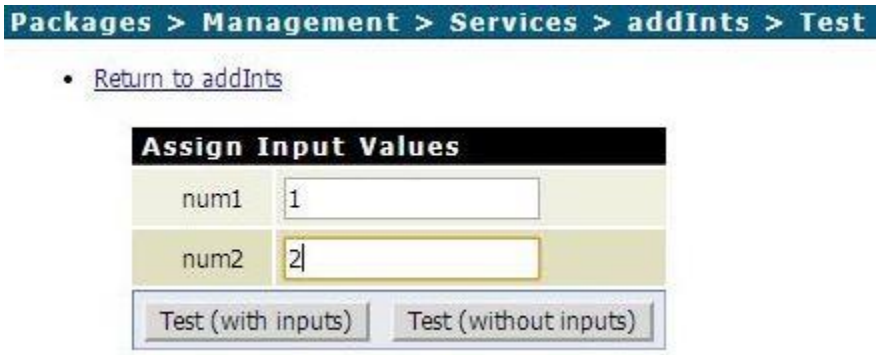
The following screen will display. The console will give you the option of testing with or without inputs.



If the Flow service requires inputs but the tester failures to enter the required inputs, then an error will be thrown:

SerrorDump	com.wm.app.b2b.server.ServiceException: [ISS.0086.9249] Missing Parameter: num1 at pub.math.addInts(math.java:405) at sun.reflect.GeneratedMethodAccessor34.invoke(Unknown Source) at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25) at java.lang.reflect.Method.invoke(M at com.wm.app.b2b.server.JavaService.baseInvoke(JavaService.java:439) at com.wm.app.b2b.server.invoke.InvokeManager.process(InvokeManager.java:643) at com.wm.app.b2b.server.util.tspace.ReservationProcessor.process(ReservationProcessor.java:46) at com.wm.app.b2b.server.invoke.StatisticsProcessor.process(StatisticsProcessor.java:44) at

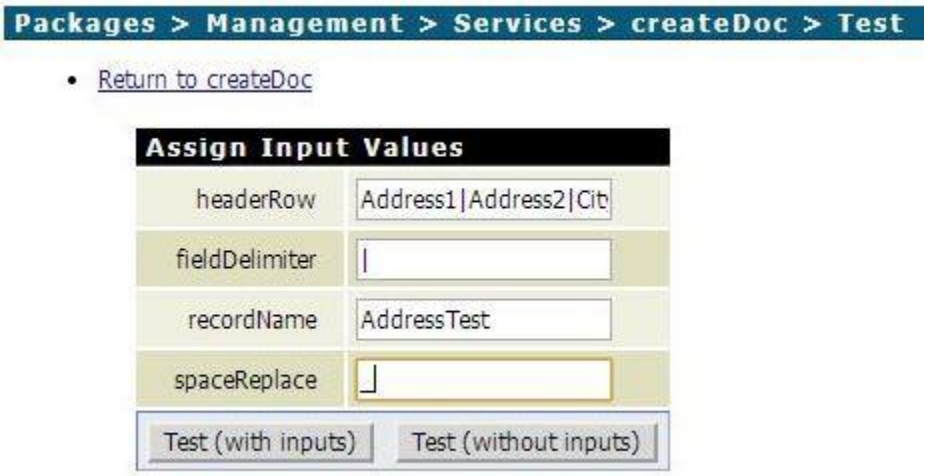
Enter the numbers 1 & 2 for the input and click on the button test with inputs:



The console will output a crude but correct HTML table with pipeline variables. The console returns exact output. If the clearPipeline command is used for the Flow, only items in the preserved array will return.

num1	1
num2	2
value	3

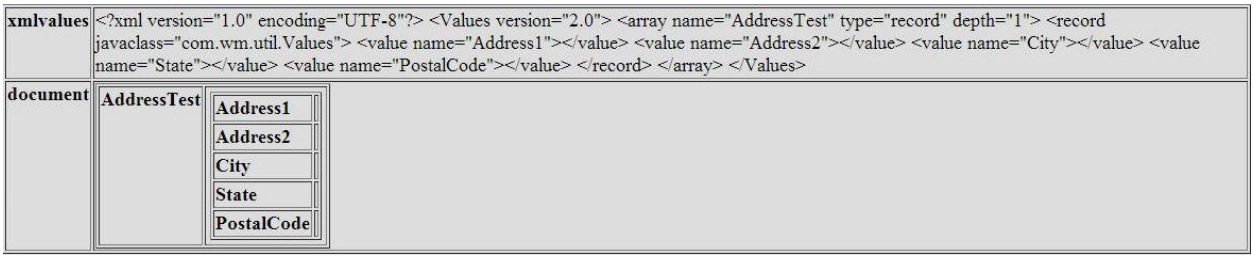
The following demonstrates how to test the Flow: cookbook.pub.adv.createDoc:createDoc.



Add in the following information and click Test with inputs button:

Input Field	Comment
headerRow	Address1|Address2|City|State|PostalCode
fieldDelimiter	|
recordName	AddressTest
spaceReplace	_

The console test results:



The "document" node is first, and AddressTest displays as the first child node.

9.4.3 USE THE URL

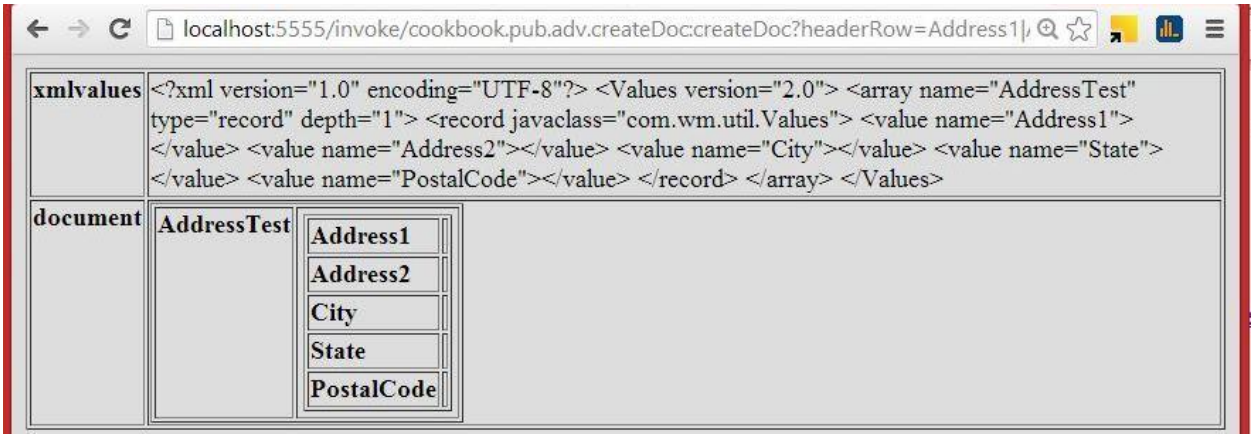
Every service component has the capability to be invoked via HTTP. The URL is typically as follows:

http://<server host name or IP>:<port>/invoke/ + servName + Params

Using the above example yields the following URL:

http://localhost:5555/invoke/cookbook.pub.adv.createDoc:createDoc?headerRow=Address1|Address2|City|State|PostalCode&fieldDelimiter=|&recordName=AddressTest&spaceReplace=_

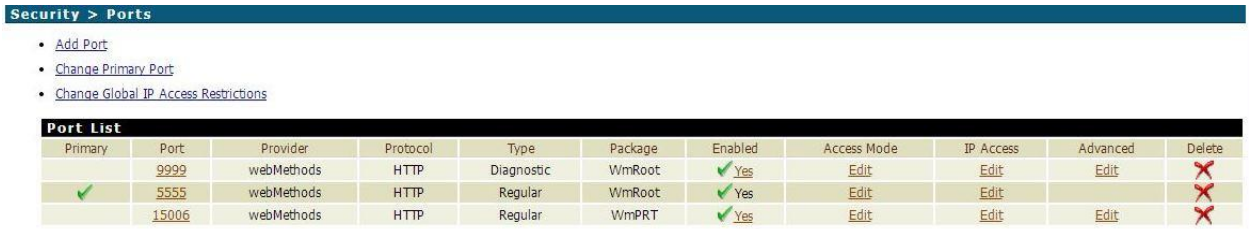
The parameters are sent using the standard URL key/value method. A question mark follows the service name and each parameter is delimited in the URL with an ampersand "&". The output of the Flow test is:



It should return the same exact information that using the console to test did. This test can be saved as a bookmark in your browser and repeated later. The URL will contain the necessary inputs.

9.4.4 Getting Data into the System – Use the Ports

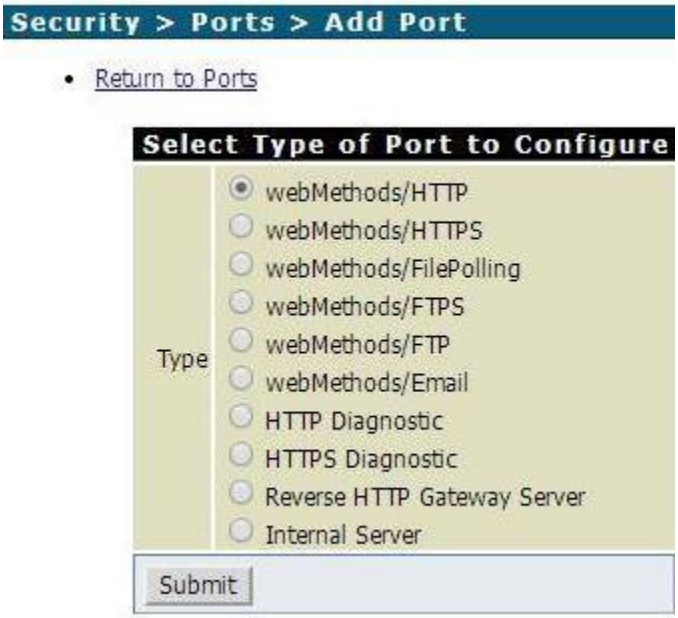
There are many ways to get data into webMethods. The following subsections detail multiple ways to submit data into webMethods. Open the Administrative console to Security > Ports.



Primary	Port	Provider	Protocol	Type	Package	Enabled	Access Mode	IP Access	Advanced	Delete
	9999	webMethods	HTTP	Diagnostic	WmRoot	✔ Yes	Edit	Edit	Edit	✘
✔	5555	webMethods	HTTP	Regular	WmRoot	✔ Yes	Edit	Edit		✘
	15006	webMethods	HTTP	Regular	WmPRT	✔ Yes	Edit	Edit	Edit	✘

Your webMethods instance may have more ports configured.

Click on the Add Port link at the top of the screen. The following radio button choice selection is displayed:



While there are many choices, the focus of this section is on only a few: FilePolling, FTP, and Email.

1. The FilePolling Port provides

s

10.0 METHODS OF DEBUGGERY

Debugging code efficiently is an art in itself. Developers must learn to debug code effectively. This primary development step insures a development or transfer of knowledge to the individual that enables them to become more efficient and then become more proficient with the webMethods platform.

In order to become more effective, the developer must develop a firm understanding of how the server works and where dependencies exist between the components.

Many overlook the capabilities of the message Broker to assist in troubleshooting over other seemingly easier but ineffective methods. An entire book could be devoted to the topic of debugging. Instead, this section provides the proven methods to quickly provide clarity into the system, how best to leverage the components, and when each method is most appropriate.

Tips and Tricks - Debugging

- Set the logging level incrementally until the right amount of information is collected.
- Choose TRACE level if all else fails, but be prepared to sift through a lot of information.
- Don't rule out crafting a custom log. This is handy if you are debugging in a system that has hundreds or even thousands of services. The server log easily gets crowded with information and it makes looking for your specific information very difficult.
- If on a production system, be sure to set the log back to the acceptable level if you made changes.

10.1 INTEGRATION SERVER

The first place to look for answers is the server logs. Early versions of webMethods wrote output to file-based logs. The majority of logging (except the server log) has migrated to the webMethods schema in the database that was set up during installation of the system. Providing that the database is up, and the network connection available, the logs can provide answers.

Note: Administrators can set properties in the Integration Server configuration properties to use log files. Use this setting if the only needs are the server and error logs. Repositories, Process Models and even some Integration Server settings find their way into a database table. It is not well documented.

10.1.1 INTEGRATION SERVER LOGS

Navigate to the Settings → Logging in the Administrative Console to display a table of logging objects. The table lists the Logger Name, Status (Enabled?), Mode, Guarantee and Destination. The destination details indicated if it is file-based (server log) or database capture.

Logger Name	Enabled	Mode	Guaranteed	Destination
Document Logger	Yes	Asynchronous	No	Database
Error Logger	Yes	Asynchronous	No	Database
Guaranteed Delivery Inbound Logger	Yes	Asynchronous	No	Database
Guaranteed Delivery Outbound Logger	Yes	Asynchronous	No	Database
Mediator Transaction Logger	No	Synchronous	No	Database
Process Logger	Yes	Asynchronous	No	Database
Security Logger	No	Asynchronous	No	Database
Server Logger	Yes	Synchronous	No	File
Service Logger	Yes	Asynchronous	No	Database
Session Logger	Yes	Asynchronous	No	Database

10.1.2 Server Log Settings

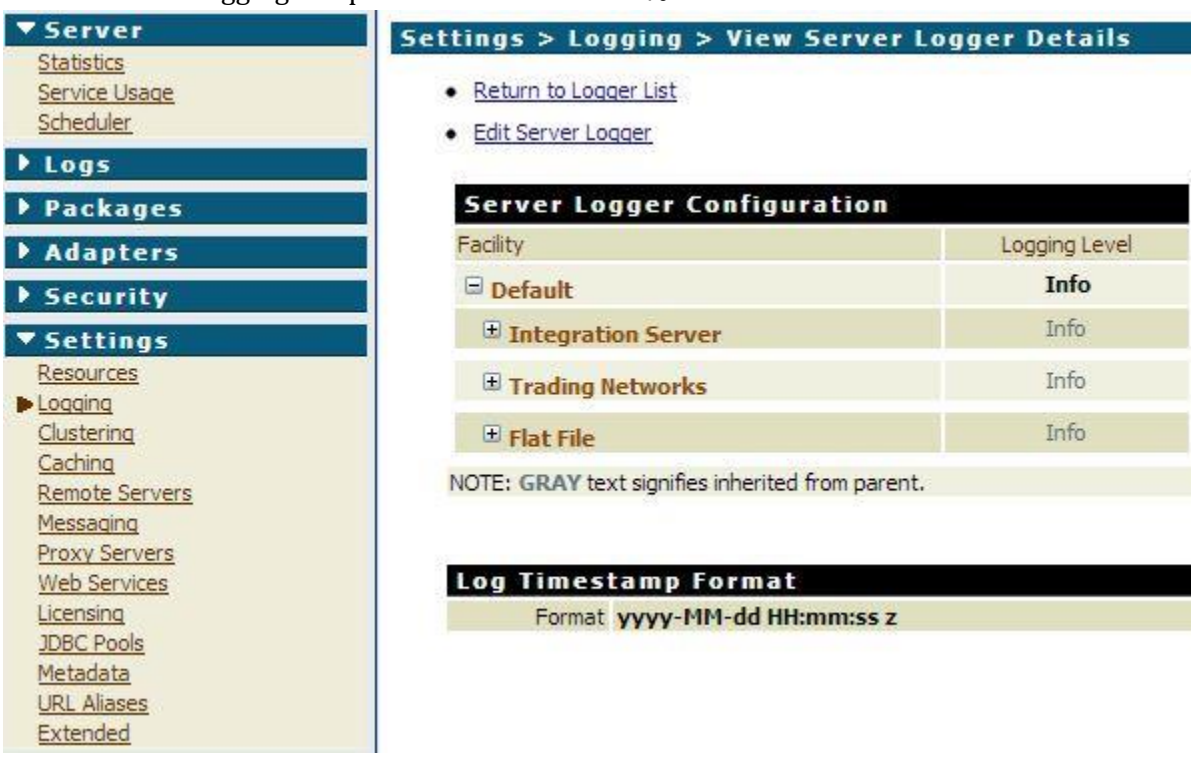
One of the most overlooked settings varies the amount of data capture in server logs. By default, Production Systems are set to capture either FATAL or ERROR messages to the server log. Development systems are typically set to capture DEBUG entries, but not always. The first place to check for information is the server log. If the system does not capture enough information, the next step is to check the logging level for the component and increase the level of capture.

To check the logging level, click on the link containing the log of interest. Clicking on the Server Logger reveals a screen that facilitates logging levels. By default, logging levels are typically set at INFO. Loggers provide these logging levels:

- OFF – Turns off logging entirely for that logger.
- FATAL – Only logs FATAL events.
- ERROR – Logs ERROR and FATAL messages
- WARN - Logs WARNING, ERROR and FATAL messages
- INFO – Logs INFORMATIONAL, WARNING, ERROR and FATAL messages
- DEBUG - Logs DEBUG, INFORMATIONAL, WARNING, ERROR and FATAL messages
- TRACE – Logs TRACE, DEBUG, INFORMATIONAL, WARNING, ERROR and FATAL messages

Click on the Edit Server Logger to change the setting. INFO provides only bare minimum information. To get more in-depth information, set the server logger to DEBUG. If this fails to provide enough information, then set the log to TRACE. TRACE provides EVERYTHING down to server level internal pings and acknowledgements. Most of this will be meaningless to someone who does not have an advanced understanding of servers, Java and TCP/IP communications. The

DEBUG level of logging will provide the answers 98% of the time.



If there is a resource concern, you may drill down into the logging facilities and manually select individual logging components for more information and leave the default setting to INFO. The sub-logging components will inherit the setting from the parent. The GRAY characters show which items are inherited. The following graphic depicts the Authentication Logger set to DEBUG. This will magnify the server events and processes that directly support the Authentication Process. With this setting, additional information above what is normally collected will be captured in the log and available for later analysis. A team that is trying to determine why Authentication is failing could use this approach to quickly obtain answers.

It is important to incrementally raise the logging level to expose more information. TRACE provides an exorbitant amount of information that is difficult to sift through, however, this is the best way to see EVERY single activity that occurs. Just be prepared to sift.

10.1.3 Server Error Log Settings

The Integration Server Error Log, by default, is set to write to the database. Sometimes, it may be less cumbersome to write to a file. By clicking on the Error Logger, you may select destination type of FILE. After a server reboot, this will write all errors to a file instead of the database. All logs can be set to write to files, but there are performance issues that can arise if the file becomes too large for the Integration Server to handle with the default settings.

10.1.4 Custom Logs

Sometimes, using the built-in server logs is not practical. It may be that read privileges are not available or the server is not generally available for troubleshooting. If a development environment serves a great number of users, the server log may become unusable or inconvenient

to use as a debugging tool. In this case, a custom log is a method to consider. A proven recommendation is to create a sub-directory called "data" under the Integration Server: ./IntegrationServer/data/. Users can write their custom logs to this directory without worrying about colliding with internal Integration Server processes.

To create and use a custom log, the follow these steps:

1. Create a Java service with Designer. Use the following information:
    - Input Signature:
        a. String: `fileContents` (information that you wish to log)
        b. String: `filePathName` (path to log file)
    - Output Signature:
        a. String: status (OK, or Error: + ErrorMessage)
    - Java Imports: `import java.io.*;`
    - Paste Java Code into Designer Editor:

```
// pipeline
// GET Input Data
IDataCursor pipelineCursor = pipeline.getCursor();
String  fileContents = IDataUtil.getString( pipelineCursor, "fileContents" );
String  filePathName = IDataUtil.getString( pipelineCursor, "filePathName" );
pipelineCursor.destroy();

String serviceMessage = "";

try {
        char buffer[] = new char[fileContents.length()]; //initialize character buffer
        fileContents.getChars(0, fileContents.length(), buffer, 0); //introduce log entry
        FileWriter fw = new FileWriter(filePathName,true); // true means to append file.
        fw.write(buffer); //write log line
        fw.close(); // close file writer
        serviceMessage = "OK"; // return status code of OK.

} catch (Exception e) {

        serviceMessage = "Error: " + e + ""; // return status code with ERROR message.
}

// pipeline
// PUT OUTPUT data into pipeline
IDataCursor pipelineCursor_1 = pipeline.getCursor();
IDataUtil.put( pipelineCursor_1, "status", serviceMessage );
pipelineCursor_1.destroy();
// END CODE
```

To test the code, click on the service and run as Java service and enter in the following information:

fileContents: `This is a test.`
filePathName: `./data/2013-07-01_Test_Log.log`

The test assumes that the "/data" directory exists under the Integration Server root. This should return:

status: `OK`

At this stage, you have created a custom logging tool that has a lot of potential uses when coupled with key utilities. For example, it could be used to debug Flow services, with one log for each Flow service. One needed utility is a file/Flow naming utility that returns the Flow name for each Flow service along with the date prefix. This will be passed into the log writer to identify the Flow service name, the date time stamp and the log message. It will be used later to help analyze the data in the pipeline to understand what is in the pipeline at a given point. Many times, developers use the same exact variables which can lead to improper results. webMethods uses Vector objects to store the pipeline at run-time. The Vector can store multiple items with the same exact name in the same pipeline. Typically, server overwrites the item with the same name but you can name objects of different type with the same name. A custom log can expose this.

Tips and Tricks – Custom Logging

- Use one log file per developer, Flow service, or other collection target.
- Nest the Flow in a branch statement that can be set to turn on/off debugging; Branch off of debug=true; If code is promoted without removing the entries, the log will not collect data.
- Use in conjunction with the Broker. Create a log collector message and publish to Broker. Create a Flow that takes log entry as input. Create a Trigger to subscribe to log collector message that invokes the custom log service. This separates the logging activity from the IS and allows for logging in a high volume, high transaction environment with less overhead.

When there are many environment users, custom logs can reduce the amount of chaos, misreads and other human error factors.

To use your new Flow service, invoke the custom log Flow anywhere it is needed to capture data. Property files may be used to indicate log file names, or other referenced inputs. These components should be part of the bigger architectural framework and the decision to use a particular pattern should be influenced by developers but approved and understood by technical leadership. This establishes an approved mechanism to use custom logging in a controlled manner.

10.2 Message Broker

The Message Broker is often the most overlooked tool. The custom logging section identified how the Broker could be used to collect log messages and process them asynchronously. To understand how to use the Message Broker requires an understanding of how it works and what role it plays in the Integration Layer and Software Stack.

The Software AG Message Broker is a legacy implementation of an EAI message Broker and is the primary Enterprise Service Bus for the product stack. Prior to the adoption of standards-based messaging, each vendor designed and built software to provide the features of an enterprise messaging bus but without committing to a standard. Several years later, Service Oriented Architecture established suggested guidelines that were later ratified and codified by the WWW. In Software AG's case, the Message Broker was extended to implement and comply with the Java Messaging Service (JMS) specification. Note that specifications do not contain instructions for

implementation. This means that every vendor implements the specification a little differently or in some cases very differently.

This can often result in "incompatibility" between vendor products that seemingly should work together seamlessly. The reason for this incompatibility is the need to use vendor-supplied client libraries to implement communications between vendor products. Vendors have nothing to gain by supplying seamless transition and translation between their products and potential competitor products.

With the history lesson complete, let's examine how the Message Broker works today. The Message Broker has morphed from simple message pickup and delivery to processing a lot of internal messages relating to auditing, reporting and task oversight.

Process Models depend on the Broker to kick off processes by using a subscription trigger and to hand off data between process model steps by using transition triggers. The Process Runtime (PRT) package facilitates, moderates and controls how the Broker is used. By gaining a greater understanding of PRT, developers can create other artifacts in the environment to understand what is happening beyond what the WmMonitor or what My WebMethods Server (MWS) displays.

For example, if a message is used to kick off a process, how can you determine if the message is published and received? One way of course is to check the queues in MWS. But what happens if the queue is empty and all you have to show for it is a failed process?

Create a Broker trigger on Integration Server. Configure the trigger to receive the SAME document as the process model. Create a Flow service that receives the Broker document and converts the document to a string, and write the contents of the message to either the server log or your custom log. Configure the trigger to call this Flow service. If the trigger picks up the document and writes the contents to the log, then check the log to insure that all of the required message fields contain data and that the data is correct. If not, look upstream from your process model to determine why the data is not correct. If the data is correct and the process model still doesn't work, then further analysis will be required at the Sub-Process model level.

The previous paragraphs detailed a common scenario and how to implement simple collection of messages. The following information will provide the components and steps necessary to implement a debugging framework that leverages the Message Broker.

1. Create a message that contains fields of value for your particular debugging. Recommended fields include:

Field Name	**Comments**
FlowName	Name of Flow service that the message is published from.
Message	Field that contains custom message for a given scenario
MessageCode	If the debugging framework uses codes to define behavior, enter the code in this field

Pipeline	Pipeline data

*Pipeline must be converted to string prior to placing in the Broker document for publication.

2. Create a Flow to capture the message. The Flow service input signature must contain a String field whose name is the fully qualified Broker document name. The Flow service should contain logic to write the information to a log, insert into a database or some other collection scheme.
3. Create a trigger that looks for the message created in step 1 and invokes the Flow service created in step 2.
4. Search the log or database (depending on your implementation) for the information required to help debug.

The Message Broker supports multiple proprietary methods

a. Publish and subscribe: Main method where a process publishes a document and the Broker makes a copy available to each and every subscriber. After all subscribers receive their copy, the Broker removes the Broker copy which completes the process.
b. Publish and deliver: A methods wherein the Message Broker delivers the message to a specified client. This is used mainly in cases where fat clients connect to the Integration Server to obtain messages. A specific client ID must be included.
c. Publish and reply: Delivers a reply document to a requestor. In this case, a Flow service could request queued documents for action.
d. There are other methods but they do not fit the pattern for use in debugging.

Tips and Tricks – Debugging with Broker

- The webMethods messaging guide discusses patterns that relate to JMS.
- Stream large files instead of loading into memory.
- Always re-sync documents with the Broker after making changes to make sure that the Broker has the most recent copy.
- A Broker trigger is a Broker client. Make sure that the Broker trigger is enabled otherwise it will not pick up documents
- Use MWS to check Broker client queues to see if the documents are queued

11.0 Error Handling Framework

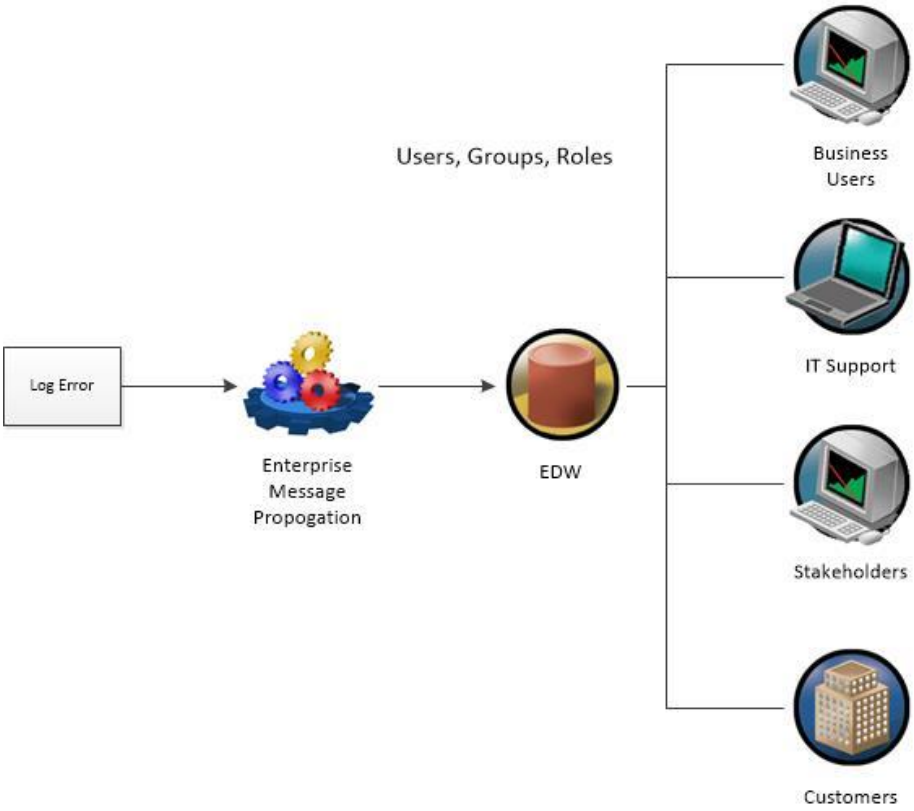
The Software AG, webMethods platform provides granular auditing and logging. With proper configuration, the system can collect the needed data.

11.1 Error Handling Framework Overview

Organizations normally architect and deliver the Error Handling Frameworks (EHF) as an Enterprise Asset. The system normally aggregates and stores information in an Enterprise Data Warehouse or an Operational Data Store based on business requirements and operational need.

Administrators assign users, groups and roles with access to the data. This will enable business owners, developers, testers and other stakeholders access to this data to help make future decisions and or to solve problems.

The EHF can aggregate error handling for more than just the webMethods stack. The error should be logged using a centralized error handling mechanism. Each application can still maintain a separate log, especially if that application depends on organic processes. However, in almost every case of modern technology in a Service Oriented Architecture, the application stack can access a web service that can consume the error message using a "log error" method.



The graphic depicts the typical Flow of data from the point of origin (log error) to the hands of the consumer. What is not depicted above is exactly how the user will access the data. The following represent the most common ways to retrieve and display the data:

- A web-based interface
- Reporting interface software such as Crystal reports
- Business Intelligence software such as Cognos
- Custom Report extracts (PDF files emailed nightly using iText)

The architecture and implementation of the methods is beyond the scope of this document.

11.2 Necessary Steps to Implement IS Flow/Java Service Error Handling and Auditing

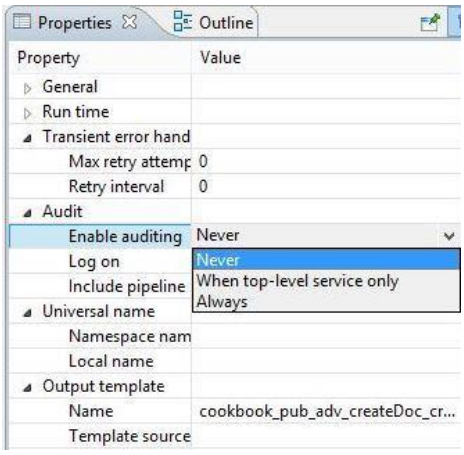
Flow and Java services represent the lowest level of processing for the platform.

The majority of the time, the system suppresses error message generation and logging. Section 10 addressed issues of debugging and noted how to set up logging to provide meaningful feedback. Section 8 introduced Flow Service custom logging.

The following pages contain the necessary steps required to enable auditing on a per-service basis. Auditing requires significant resource allocations and can severely degrade performance.

Follow these steps to enable auditing and logging at the Flow Service level:

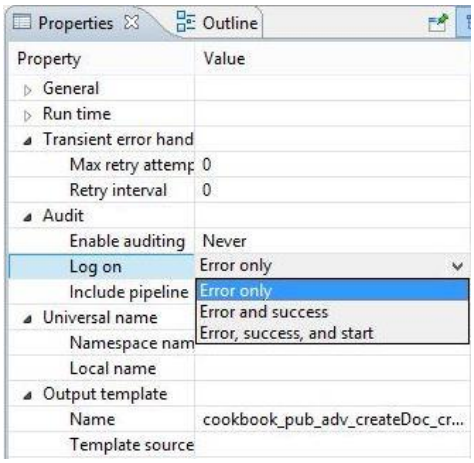
1. Open up the Flow Service Properties for the target Flow and set up the auditing processes for logging. The Audit property contains 3 potential settings.



Enable Auditing
To enable auditing for a given Flow, find the "Enable auditing" setting. Choose between:

- Never – Never audit this Flow service.
- When top-level service only – Only applies to Parent Flow services.
- Always – Will always audit.

2. Setting the available logging properties.

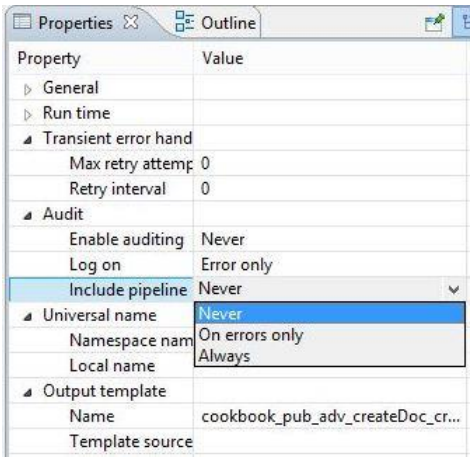


Log On Property

Find the "Log on" properties:

- Error only – will only log if an error occurs
- Error and success – logs the 2 potential statuses, error or success
- Error, success, and start – logs error, success and start. This setting provides full scrutiny.

3. Determine when and if the pipeline should be captured.



Include Pipeline

Find the "include pipeline" properties setting:

- Never – Never save a pipeline
- On errors only – If an error occurs, will save the pipeline
- Always – Will always save the pipeline. Highest scrutiny.

12.0 TESTING

12.1 UNIT TESTING

The unit test will specify testing parameters along with success/failure conditions that must be examined. The developer assigned to a particular section has responsibility to test each piece of the Integration.

The developer should write a test plan and submit the test plan for approval prior to developing the code. The test plan must contain traceability to requirements. Just as an integration project relates to a business case, the test plan must contain the parameters necessary to confirm that the integration project or code portion answers the business case in total. In addition to the business case requirement, error trapping, debugging and other architectural items will be checked.

The developer will perform the tests in the appropriate environment after coordinating with other team members. Once the tests have been performed satisfactorily, then the code is signed-off by the developer and accepted by the build manager, and the documentation is notated.

Unit testing is mandatory for each part of the code. Instances where code is shared between NEW packages, the unit tested code cannot be released until all the code passes unit testing.

Unit testing will be conducted using local tools such as webMethods Designer, a testing mechanism coded in Flow code, a client, SOAP UI or an external application capable of auto-testing, such as Load Runner.

Note: Specific examples to support unit testing are located in section 9.

12.2 SYSTEM TESTING

Prior to a coordinated movement of code from one environment to another (a major project release), an entire end-to-end system test should occur. Each part of the Integration should be tested according to the System Testing Approach and the individual System Testing Use Cases.

System Testing is an inevitable part of the journey from Development to Production. It represents the first time code has been migrated into a "joint" environment. It is this environment that tests if the new code works together with existing code. Analysis of the results will determine if the test run was successful. If successful, then the next steps require the documentation of the test results and submission to the Integration Testing environment.

If unsuccessful, the analysis may reveal issues immediately or may require deeper debugging. The build manager must take care to insure that the environment is configured properly to support the new code. This may include setting up new JDBC Adapter connections, installing Broker components or other new configurations. Review everything. Make sure that the code has the correct permissions in the new environment.

It could be that the new code set negatively influenced the existing code set or existing code may affect the new code. This could simply be resource sharing issues.

Continuity is the key to success. Using repeatable test cases with verifiable outcomes should be the primary focus.

12.3 Integration Testing and User Acceptance Testing (UAT)

User Acceptance Testing provides customers with the opportunity to use a production-configured system to test functionality without affecting real systems. The idea is to give customers a playground to test functionality and sign-off upon satisfactory testing. UAT is not regression or performance testing. It is testing to confirm that data formats are correct, web functionality, including cosmetic and validations are working properly, and the solution meets or exceeds requirements.

13.0 JAVA DATABASE CONNECTIVITY (JDBC)

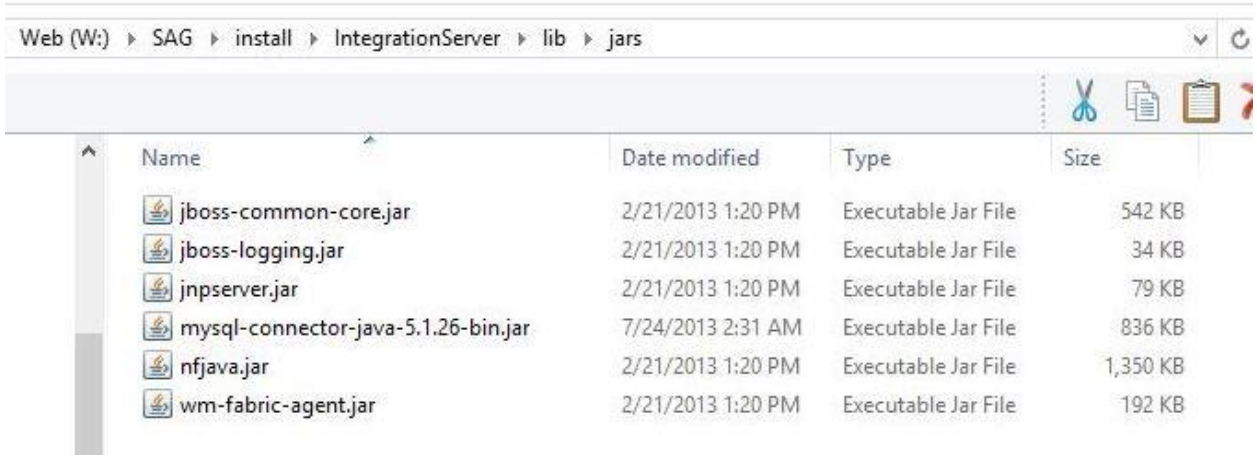
The webMethods Integration Server provides two methods of accessing database systems. The WmDB package and the JDBC Adapter. The following sections provide information and details for using each.

13.1 WMDB – WEBMETHODS LEGACY DATABASE PACKAGE

The WmDB package must be installed on the IS server. WmDB was the original package that provided DB connectivity for the platform. Version 6.x of the webMethods platform introduced the JDBC Adapter and adoption followed soon after. Original plans called to deprecate the WmDB package, however, customer requests to retain the package over whelmed the company and they decided to continue to maintain the code set.

The WmDB package makes it very easy to instantly connect to a database and commence database operations. The package supports transactional and non-transactional methods. There is no need to create an adapter connection, because the WmDB package is not an adapter and does not follow the pattern that an adapter does. For the purpose of this section, MYSQL version 5.6.x was used. The following information is specific for open-source MYSQL.

Prior to using any database, install the required java "mysql-connector-java-5.1.26-bin.jar"file in the Integration Server root/lib/jars directory:



Restart the server for the change to be effective. Other database drivers may require a certain version of a driver (Oracle, DB2, etc.) depending on the JVM version. To make sure the new driver is in the classpath, be sure to click on the "ABOUT" link on the IS Administration Console and search for the driver file name.

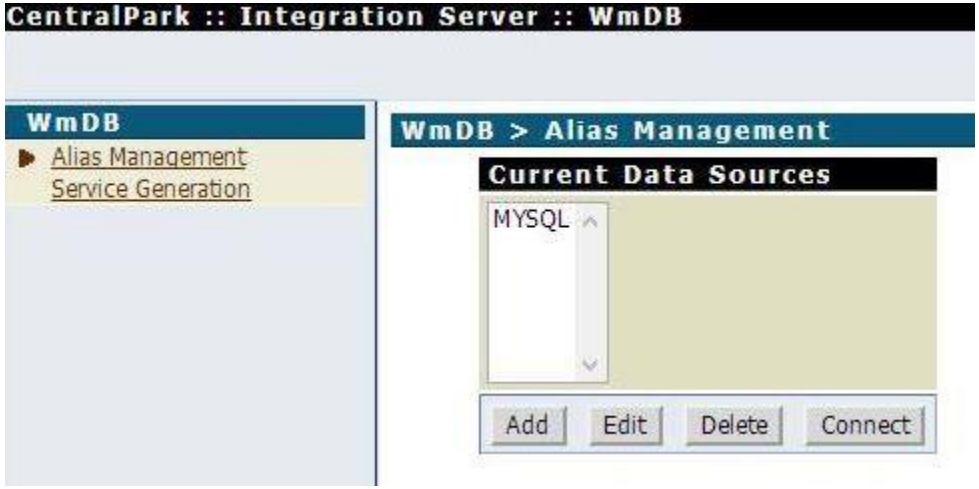
WmDB – Administrative GUI

Assuming that the database is installed and the JDBC connectivity is in place, it is very simple to connect to the database and commence operations.

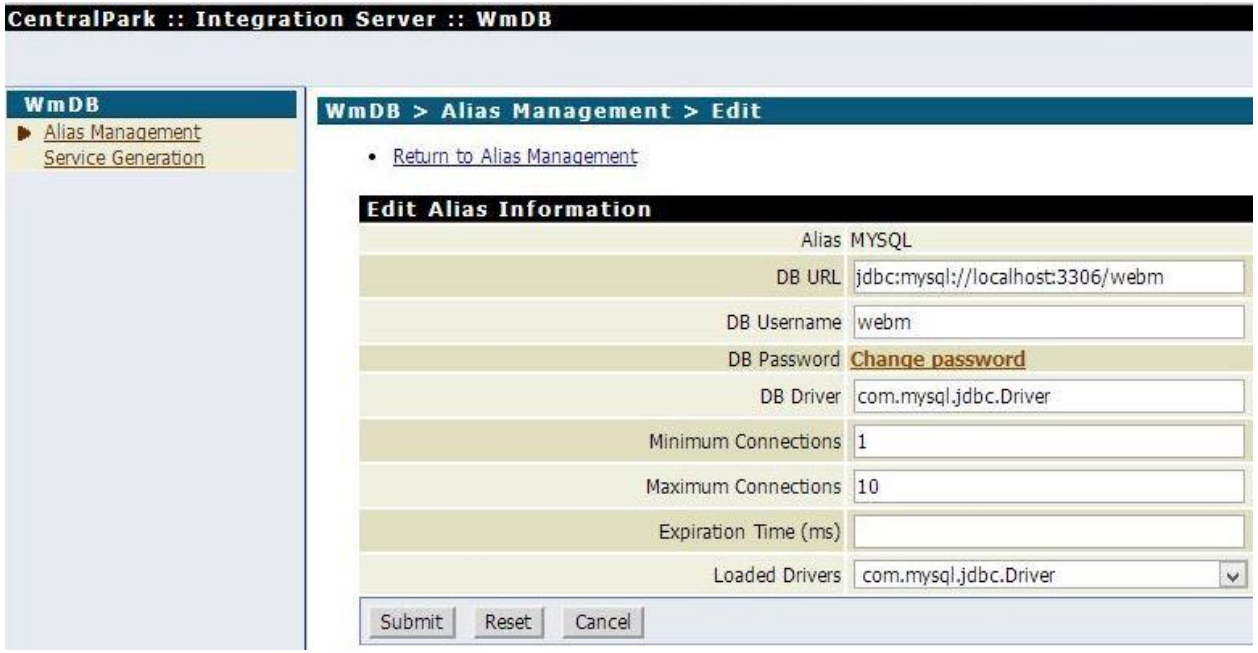
There are four variable values needed to connect to a database:

Variable	Value
$dbUser	webm
$dbPass	webm
$dbURL	Jdbc:mysql://localhost:3306/webm
$dbDriver	com.mysql.jdbc.Driver

Navigate to the Adapter menu on the left side of the Administrative console and choose the "WmDB" link. A window will open up. WmDB supports the use of Alias objects. Developers can associate DB connections to Alias objects which abstracts out the DB profile information.

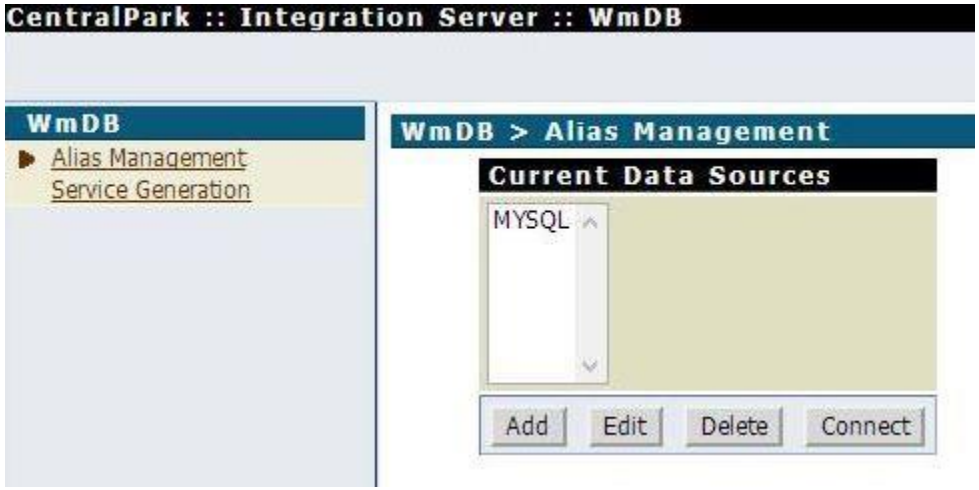


Selecting the "MYSQL" data source and hitting the "CONNECT" button provides the following information for the MYSQL alias:

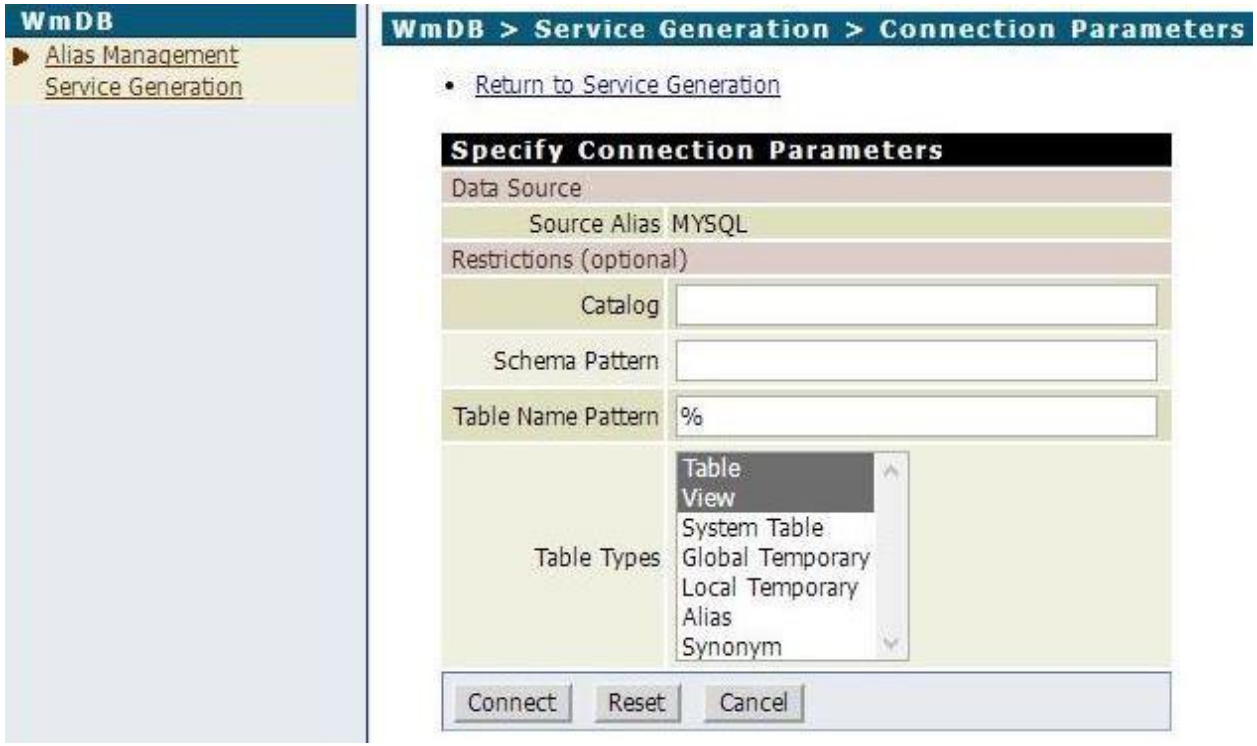


Changes can be made on this screen. The Loaded Drivers combination box contains ALL loaded drivers and permits switching between Drivers. The $dbAlias refers to MYSQL. Developers can use the $dbAlias instead of inputting all of this information for every connection in flow code.

Click on Alias Management, select the MYSQL alias, and hit the connect button:



The following screen should appear:



By default, both Table and View types are selected. To filter, add filters for Catalog, Schema Pattern or Table Name Pattern. For large databases, this makes it simple. Hitting the CONNECT button reveals the following tables for Schema "WEBM" on the following screen:

CentralPark :: Integration Server :: WmDB

WmDB
- Alias Management
- Service Generation

WmDB > Service Generation > Connection Parameters > Tables

- Return to Service Generation
- Return to Connection Parameters

Tables in MYSQL

Name	Type	Catalog	Schema	
veh_airplanes_tbl	TABLE	webm		
veh_cars_tbl	TABLE	webm		

Test SQL Queries

Execute query

The tutorial database contains several tables. Selecting the "veh_airplances_tbl" link will display the following screen:

CentralPark :: Integration Server :: WmDB

WmDB
- Alias Management
- Service Generation

WmDB > Service Generation > Connection Parameters > Tables > Columns

- Return to Service Generation
- Return to Tables

Service Type (select columns below)

Select	SELECT * FROM veh_airplanes_tbl WHERE col1=? AND col2=? AND ...
Insert	INSERT INTO veh_airplanes_tbl (col1,col2,...) VALUES (?,?,...)
Delete	DELETE FROM veh_airplanes_tbl WHERE col1=? AND col2=? AND ...
Update	UPDATE veh_airplanes_tbl SET dest1=?, dest2=?, ... WHERE targ1=? AND targ2=? AND ...

Columns in veh_airplanes_tbl

Use this column	Name	Data type	Nullable?
	veh_airplane_id	INT	NO
	airplane_make	VARCHAR	NO
	airplane_tail_num	VARCHAR	NO
	submission_date	DATE	YES

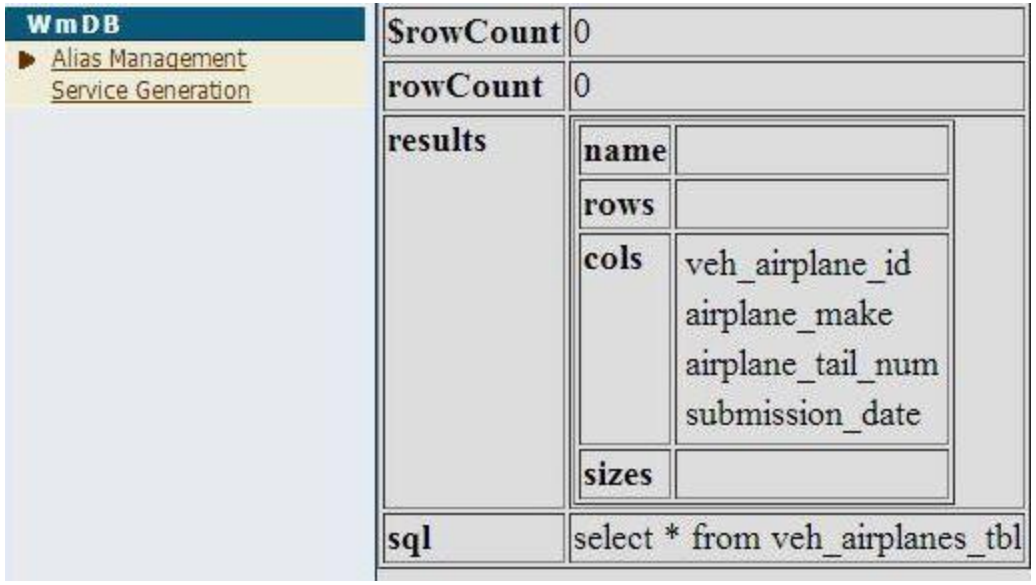
Generate SQL

Test SQL Queries

select * from veh_airplanes_tbl

Execute query

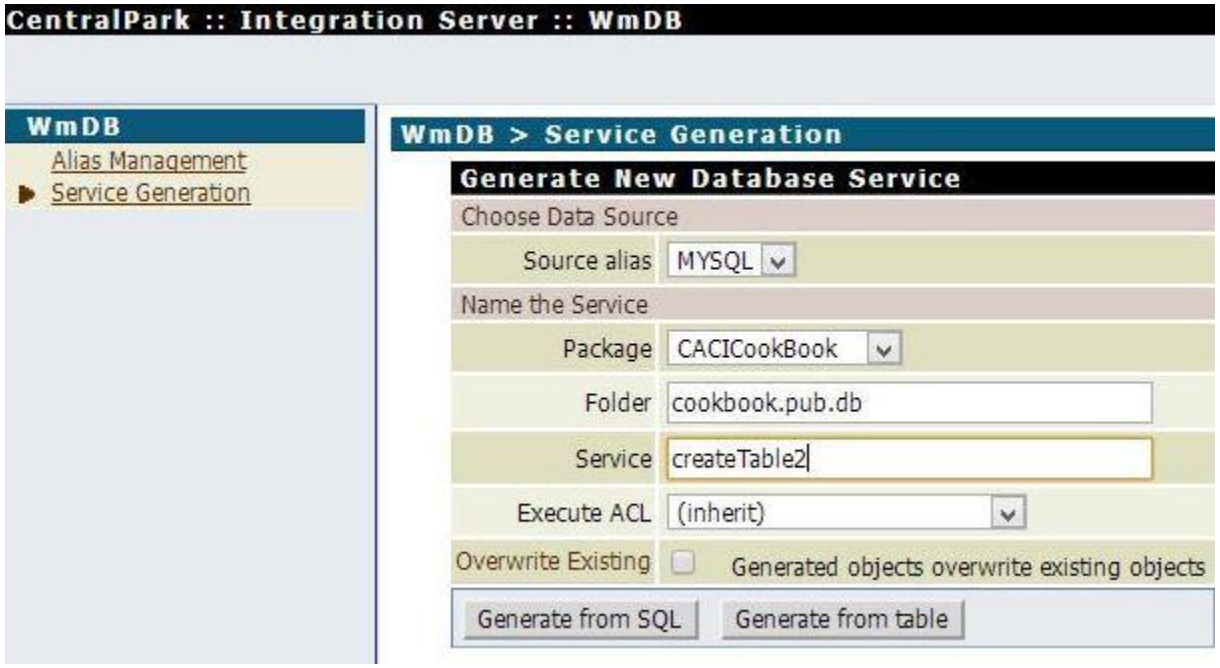
The box below "Test SQL Queries" has a select all statement. Click on Execute Query. The table contains zero rows, however, the adapter provides feedback:



NOTE: There are many other methods that can be used to manipulate data from a datasource. This includes select, insert, update, and delete methods. The WmDB package supports transactional and non-transactional executions. It also supports stored-procedures.

Developers can generate SQL-executable flow services from the WmDB administrative console. Click on the "Service Generation" link on the left and fill out the needed information:

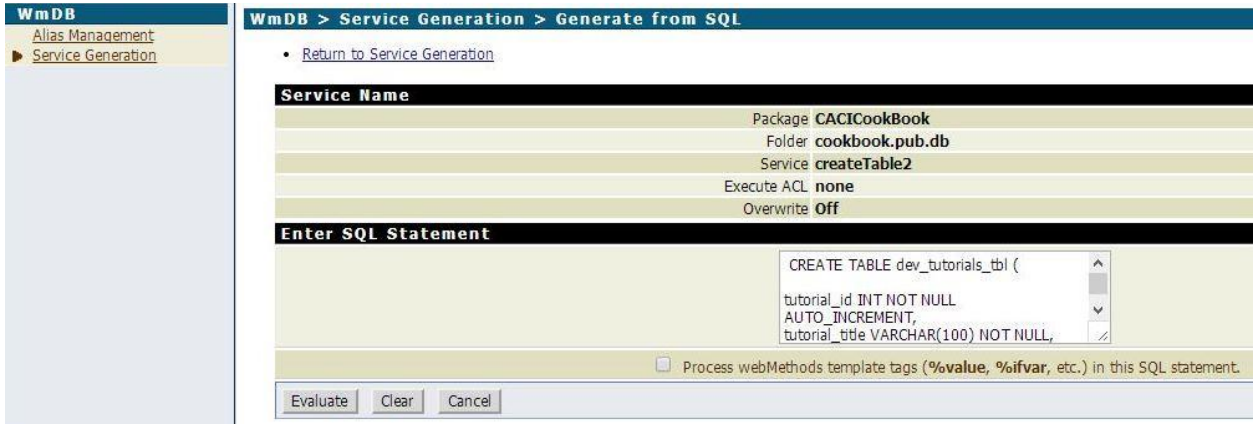
Variable	Value
Source alias	MYSQL
Package	CACICookBook
Folder	cookbook.pub.db
Service	createTable2



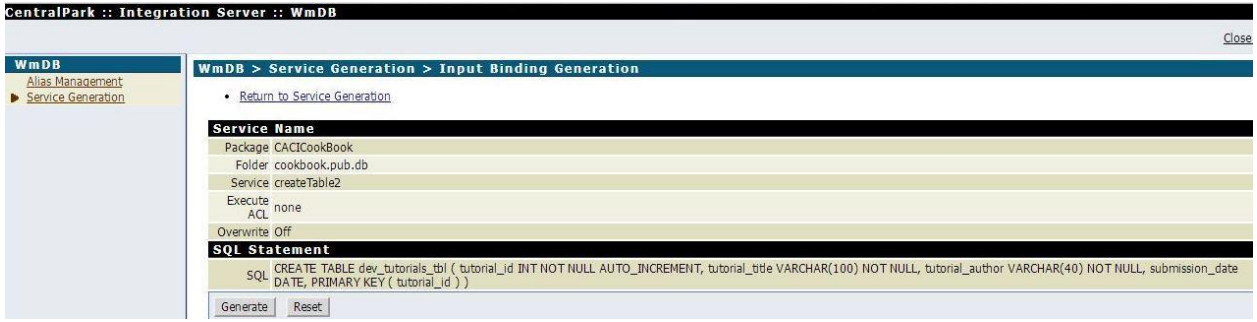
Click on the "Generate from SQL" button.

Populate the SQL Statement box with a create table statement:

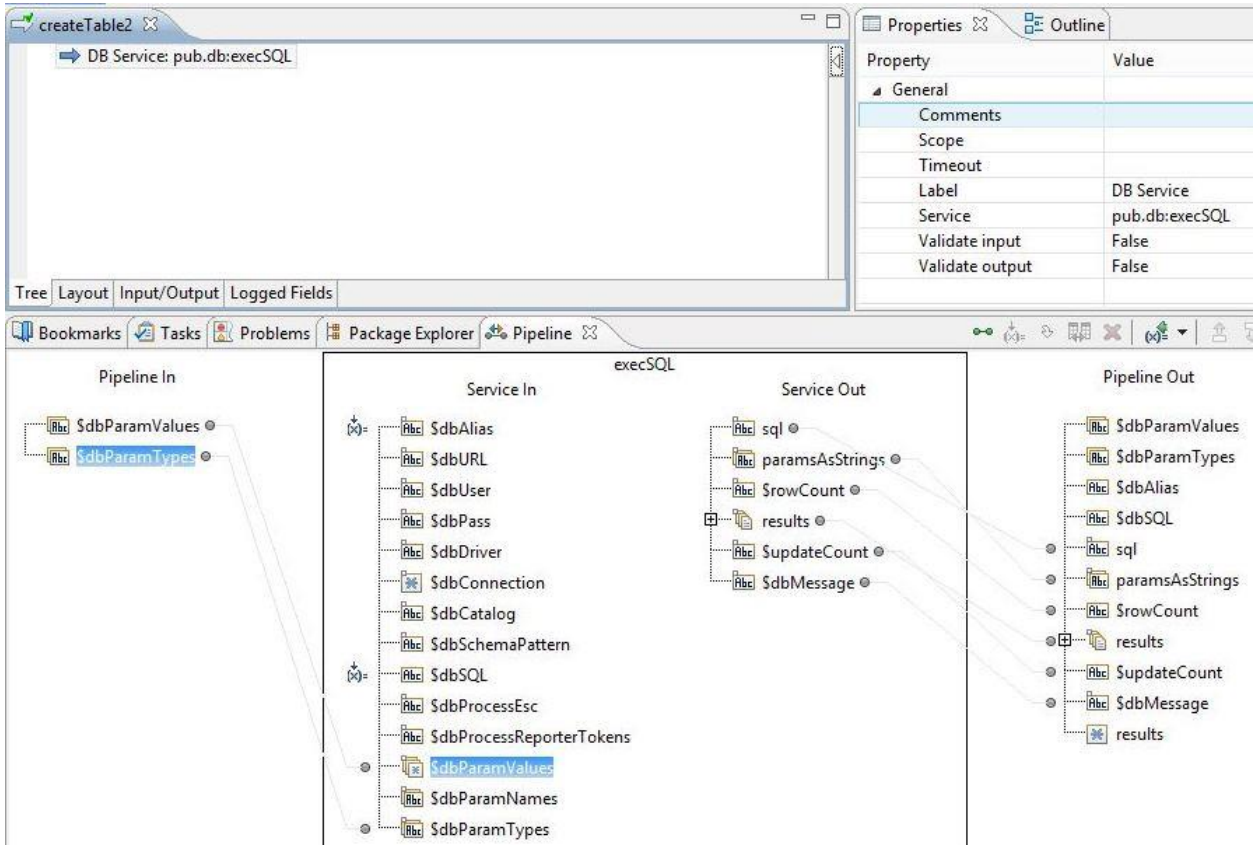
```
CREATE TABLE dev_tutorials_tbl (
      tutorial_id INT NOT NULL AUTO_INCREMENT,
      tutorial_title VARCHAR(100) NOT NULL,
      tutorial_author VARCHAR(40) NOT NULL,
      submission_date DATE,
      PRIMARY KEY ( tutorial_id )
)
```



Click on the Evaluate button. If everything looks fine, click on the GENERATE button.



The system generates a flow code with an embedded call to pub.db.execSQL:

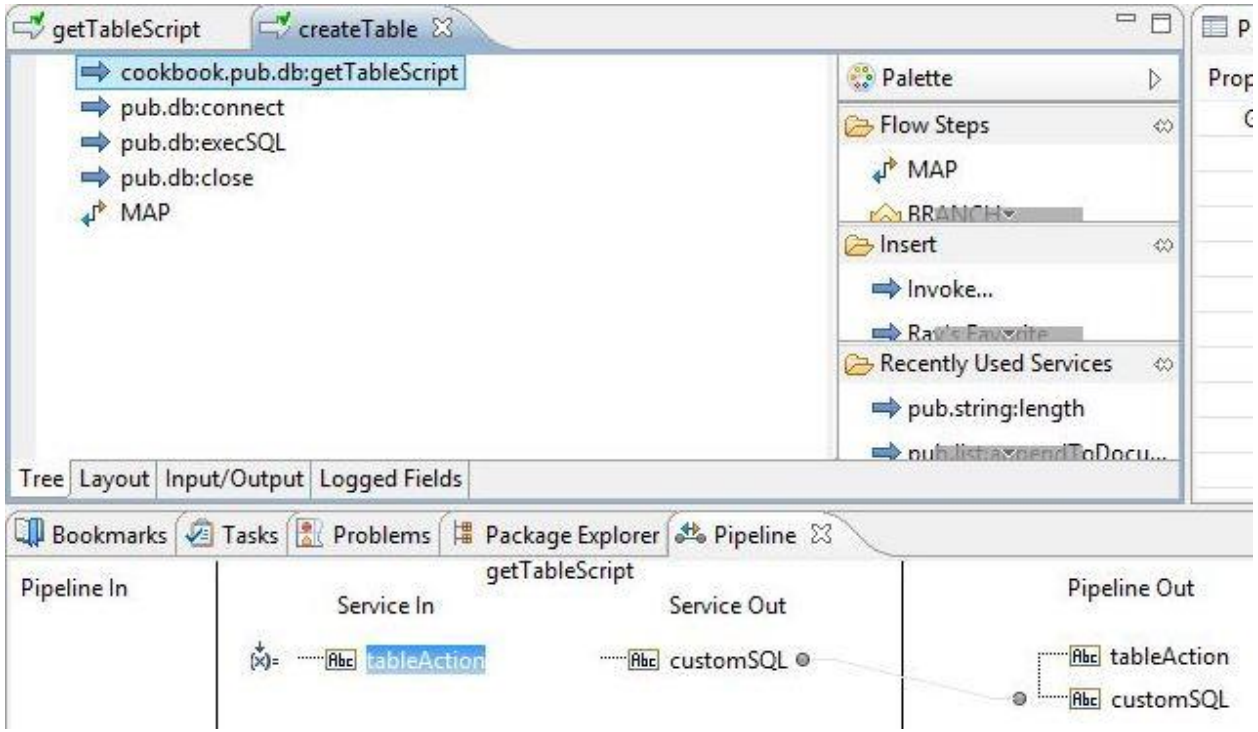


All the values are pre-set. To create the table described in the previous create table statement, run the flow service.

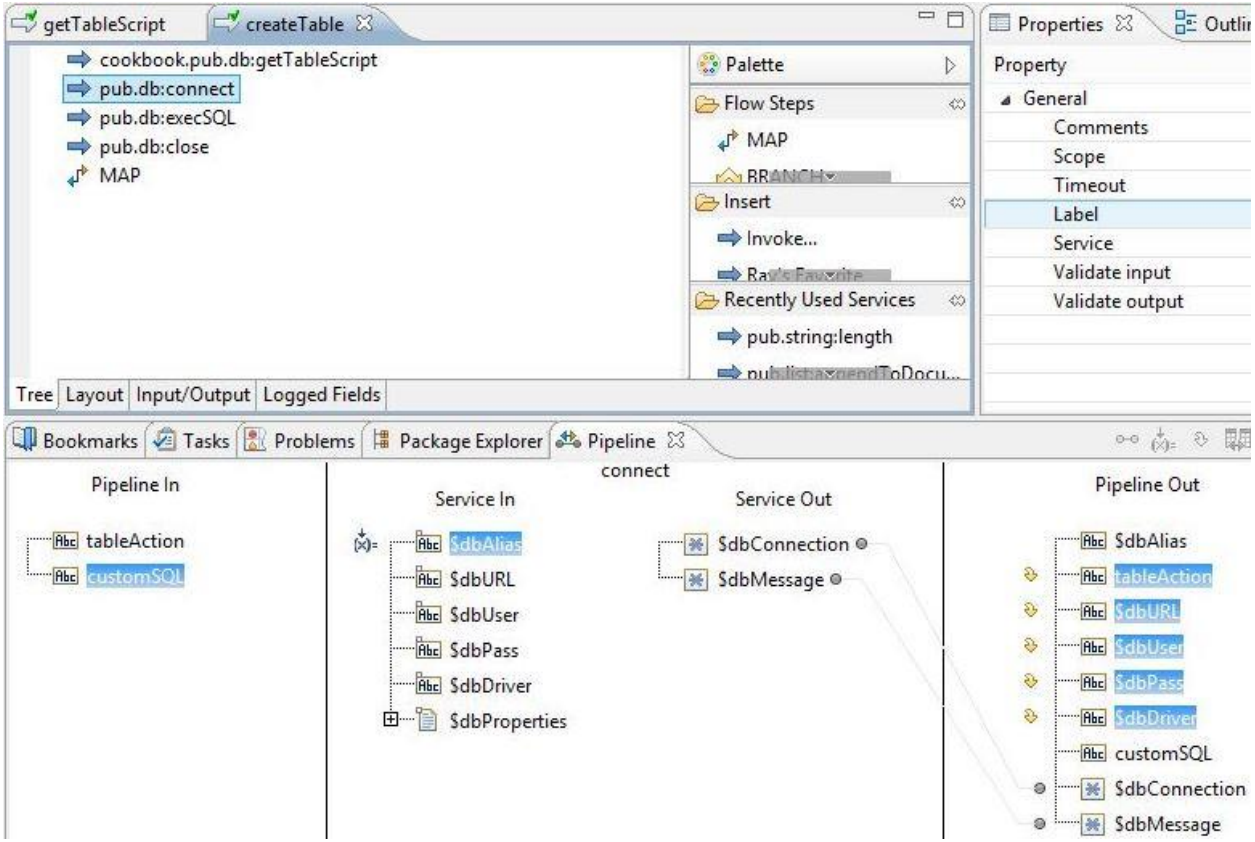
WmDB – Flow Code

Using WmDB in flow code is pretty simple. By establishing an Alias for the database, developers can quickly connect and obtain needed data or create or prototype code. It is not recommended to use WmDB in high-volume/high-transaction environments. It is recommended instead to use the JDBC adapter.

Consider the following flow:

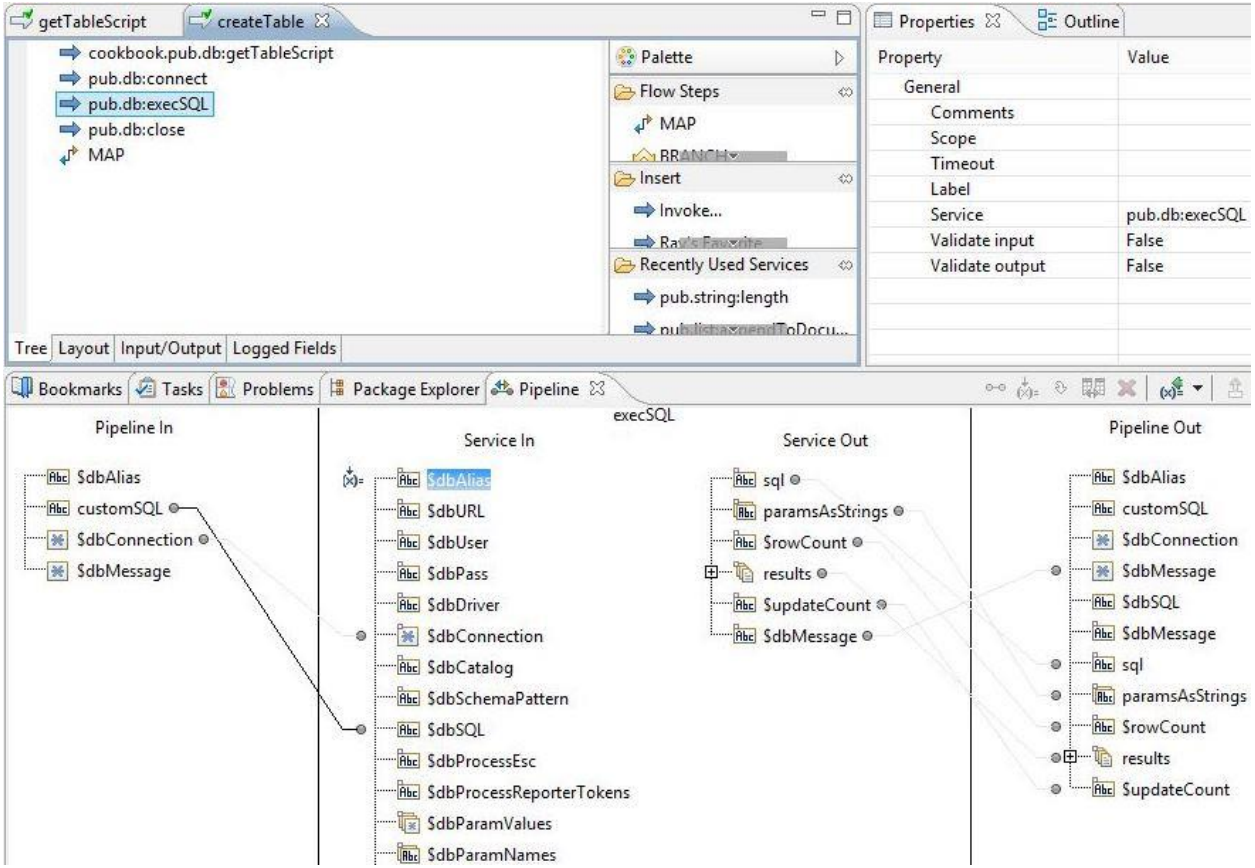


This flow service obtains a create table script, connects to the database, executes the SQL script and closes the database connection. The pattern shown, connect, <do some DB execs>, close connection represents the correct pattern. If the pub.db:close service is not called, then the connection will remain open for the user specified in the $dbAlias. The getTableScript flow requires a value for tableAction: createTable or deleteTable. It will return the required SQL statement. The next step in the flow is the DB connection:



The $dbAlias value is “MYSQL”.

Next step in the createTable script is pub.db:execSQL:

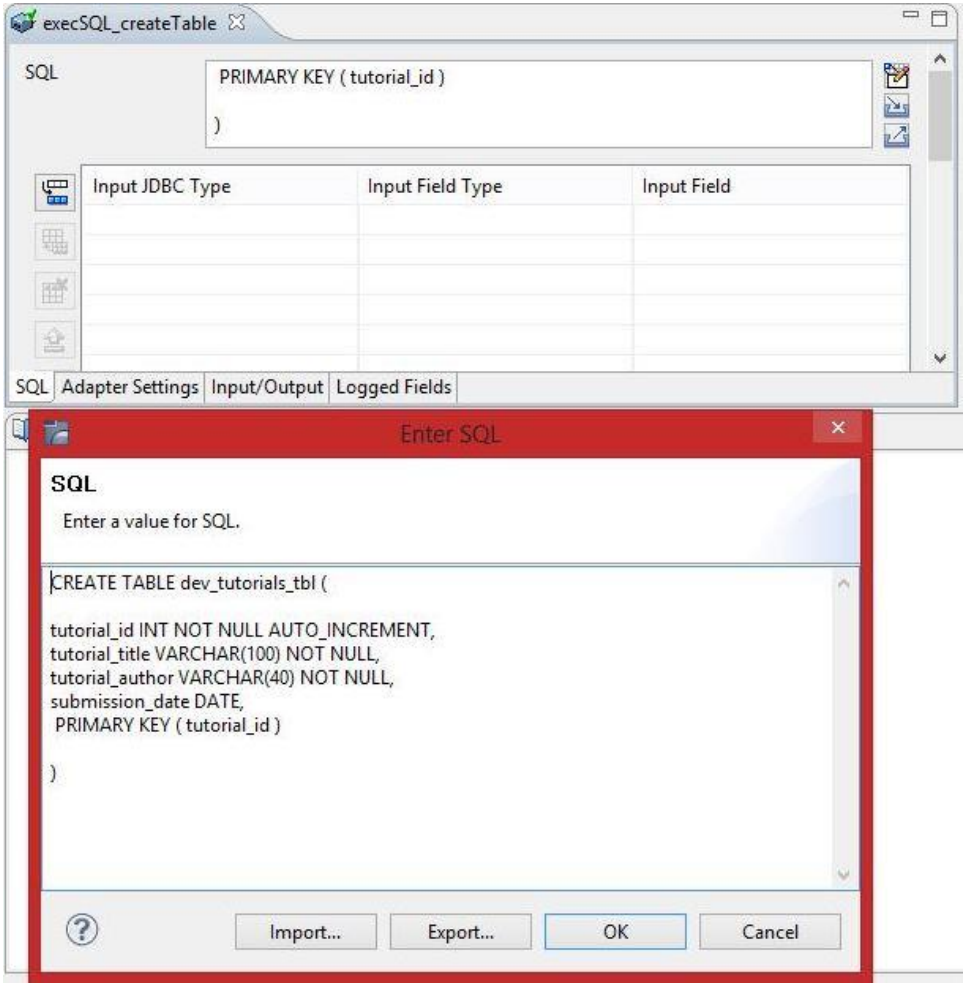


The output from the first flow provides the $dbSQL input. The next step merely closes the connection with a call to pub.db:close by passing in $dbAlias value of MYSQL.

Checking the database will confirm creation of the new table.

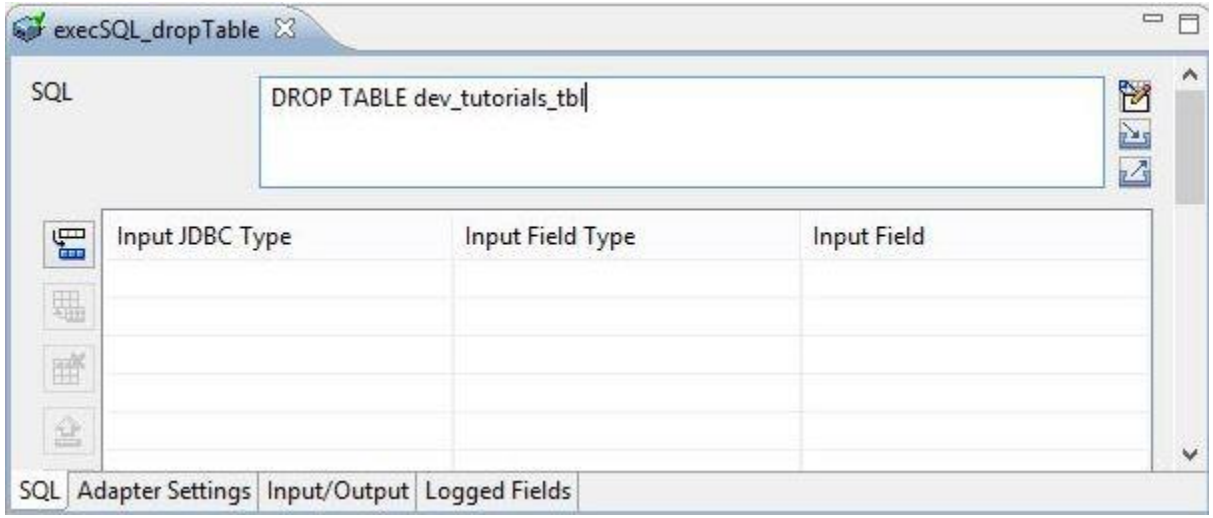
WMDB – QUICKLY CREATE OR DROP DB OBJECTS

Run as a service, pub.db:execSQL and enter in the database script.



Hit OK to create the table.

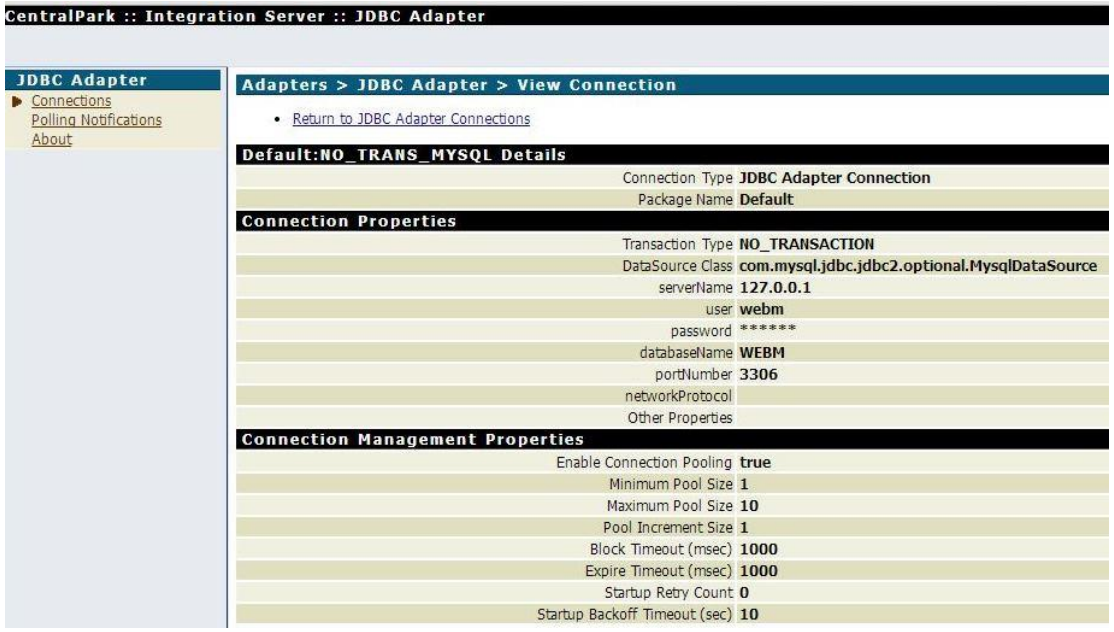
To drop a table, make the same call to pub.db:execSQL and enter in a DROP Table command and hit OK. The table will DROP.



13.2 JDBC ADAPTER

The JDBC Adapter is the preferred method to connect to datasources. There is a multitude of resources that cover the use of the JDBC adapter. Of primary important is the difference between this adapter and WmDB. Notice that either LOCAL or NO TRANSACTION type must be specified during the creation of the adapter connection. NO TRANSACTIONS do not require a start/commit/rollback. LOCAL specifies that start/commit/rollback must be used.

Typical Screen. Notice the DataSource class is different than WmDB.



14.0 Using Source Control

This section describes how to check-in and check-out code in subversion. It also details how to tag and untag code as needed.

14.1 Cooperative Development Practices

Integration Server supports project code development by multiple developers by engaging cooperative development. This provides developers with the ability to lock portions of a project to prevent others from overwriting work on the Integration Server. Developers must lock the package/service prior to beginning work. Prior to saving the work in the source control system, the developer will make sure that all locks have been unlocked. Review the Designer help guide for further information.

14.2 Subversion Change Management System

Subversion is an open source software source code "Change Management" (CM) system for collaborative development. It maintains a history of file and directory versions. The files and directories are checked out of the repository and into your local project work area. This called your "working directory". Changes are made to files in your "working directory". After changes are made to the create the next working version, the files are checked into the Subversion CM repository.

See APPENDIX C for a list of the most common SVN commands.

14.3 GIT Change Management System

GIT is a free and open source, distributed version control system designed to handle small to very large projects with speed and efficiency.

14.3.1 GIT Best Practices

Commit Related Changes

A commit should be a wrapper for related changes. For example, fixing two different bugs should produce two separate commits. Small commits make it easier for developers to understand the changes and roll them back if something went wrong.

Test Code Before You Commit

Resist committing something to the repository until it is thoroughly tested. This will keep bugs from propagating from subsequent code check out by others.

Use Branches

Branching is one of GIT's most powerful features and was a major driver in the design. Branches are a perfect tool to help avoid mixing up different lines of development. Use branches extensively in the development workflows: For new features, bug fixes, ideas, etc.

Commit Often

Committing often keeps commits small and helps to focus the commits on specific changes. It's easier to share relevant code and makes it easier to merge code later. Having few commits and rarely sharing them makes it hard to solve conflicts.

Write Good Commit Messages

Begin the message with a short summary of the changes (no more than 50 chars). Separate it from the following body by adding a space in the end (think of it as an email subject line). The body of the message should provide detailed answers to the following questions:

1. What was the reason for the change
2. How does it differ from the previous implementation?

Use the imperative, present tense (change, not *changed* or *changes*) to be consistent with generated messages from commands like "`git merge`".

Don't Commit Half-done Work

You should only commit code when it is completed. This doesn't mean that you have to complete an entire large feature set before committing. Instead, split the feature's implementation into logical chunks and remember to commit early and often. Don't commit for the sake of committing or just to have something in GIT before leaving the office at the end of the day. Instead, consider using the "`git stash`" feature instead.

14.3.2 Basic GIT Workflow Example (from salesforce.com)

To begin using GIT, move all the files into a directory (that is planned for continued use) and commit an initial snapshot:

```
$ git init
$ git add
$ git commit –m 'initial commit'
```

Create a new branch called featureA, then check it out so it is the active branch. Then, edit and stage some files and finally commit the new snapshot.

```
$ git branch featureA
$ git checkout featureA
$ (edit files, do the work)
$ git add (files)
$ git commit –m 'add feature A'
```

Switch back to the master branch, revert the featureA changes you just made, then edit some files and commit the new changes directly in the master branch context.

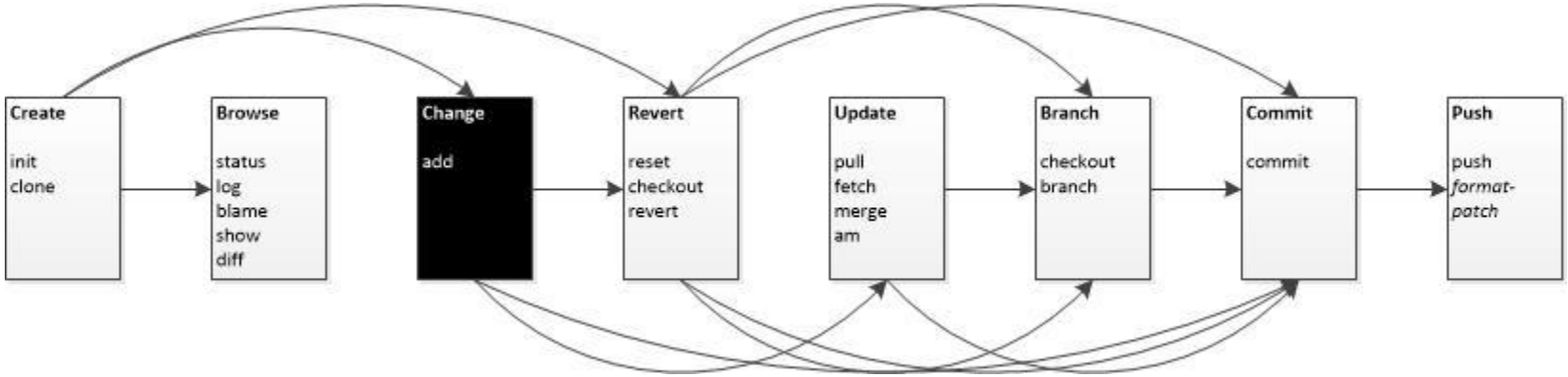
```
$ git checkout master
$ (edit files, do the work)
$ git commit –a –m 'change files'
```

Merge featureA changes into the master branch context, combining all of your work. Finally delete the featureA branch.

```
$ git merge featureA
$ git branch -d featureA
```

See APPENDIX D for a list of the most common GIT commands.

14.3.3 GIT Cheat Sheet Command Flow (Left to Right)



Basic Commands for GIT:

Use git help *command* if you are stuck.

GIT COMMAND	Description
`master`	Default level branch
`origin`	Default upstream branch
`HEAD`	Current branch
`HEAD^`	Parent of HEAD
`HEAD~4`	Great-great grandparent of HEAD
`foo..bar`	From branch *foo* to branch *bar*

15.0 DEPLOYING YOUR CODE

The purpose of this section is to describe the deployment conventions that will govern webMethods project deployment. It should serve as a guide to facilitate consistent deployment and completeness of activity. It sets standard procedures for naming as well as tagging builds for deployment. All webMethods project deployments can be performed manually. These manual deployments may also be automated using a tool like Jenkins in combination with the Deployer. This section covers both types of deployments where possible.

15.1 OVERVIEW OF WEBMETHODS DEPLOYMENT

While the majority of webMethods project development occurs using either the Developer or Designer tool, the underlying objects are created and stored on the Integration Server. webMethods IS/developer was one of the first 4GL tools to provide drag-and-drop coding. This abstraction from physical coding to 4GL coding removes the complexity with the integration of message buses, adapters, JMS services, etc. More often than not, there are complex dependencies on other webMethods components that must be deployed and configured before the webMethods project can execute successfully in the target environment. A typical webMethods project can consist of various components, thus a successful deployment should take into account all of these components:

- Integration Server packages
- Integration Server configuration
- Broker Assets
- BPM Models
- Property Files
- External Dependencies (such as database changes or connections to external systems.)

15.2 STANDARD DEPLOYMENT

15.2.1 INTEGRATION SERVER

This section provides information related to the deployment of Integration Server packages.

Step	Man.	Auto	Step Information
1	x	x	Check in code into the designated source control system.
2	x	x	Determine if any Integration Server configuration changes are needed for the webMethods code to work in the target system. This may include new users, groups, ACLs, etc. These type of configurations are specific to each Integration Server run-time environment and are not part of the code.
3	x		Document the configuration required in the target Integration Server environment to match the configuration changes noted in the deployment guide that are required to support the code.
4		x	On the source Integration Server, navigate to the following directory: `<wm_install>IntegrationServer/config/` and identify all affected configuration files (*.cnf) (i.e., users.cnf, ports.cnf, etc.)

5		x	Check into source control the configuration file identified in the previous step into the /config folder for the given Integration Server package. <package_name>/config
6		x	Remember to include only configurations that are specifically affected by the deployment. The target IS will add everything it finds in this directory so do not add or send extraneous configurations. For example, the users.cnf file should only include user details required to be pushed to the target server. Do not include users defined for a given environment such as development or test user accounts.
7		x	

15.2.2 Broker Assets

This section sets the guidelines for deploying Broker Assets.

Step	Man.	Auto	Step Information
1	x	x	Determine if any Broker assets are needed for the webMethods code to work. This may include document types, client groups, etc.
2		x	Document steps to add the Broker assets through My WebMethods Server in the target environment in a deployment guide.
3		x	On the source My WebMethods Server, navigate to `Applications` → `Administration` → `Messaging` → `Broker Servers` → `Brokers`
4		x	Click the `Broker Name` hyperlink
5		x	Select the `Asset Management` Tab
6		x	Select the assets that are affected by the deployment and then click the `Add to Export` button.
7		x	Review the selections and click the `Export Broker Assets` button.
8		x	Unzip the file and check-in the XML file into the designated source control system under <package_name>/resources/Broker
9		x	Check to insure that the xml file follows the `Broker_<package_name>.xml` naming convention.

15.2.3 BPM Models

This section sets the guidelines for deploying BPM model artifacts.

Step	Man.	Auto	Step Information
1	x		Document steps to add the process model using the Software AG Designer to the target environment in a deployment guide.
2		x	Open the Software AG Designer to a workspace that contains the BPM project.
3		x	Click `File` → `Export..` → `General` → `File System`
4		x	Select the BPM Project and export it to a local destination
5		x	Check-in the BPM Model (.project file) into the designated source control system. This should be checked into the following directory: `<package_name>/resources/bpm`

6		x	Check to make sure tha the directory contains the following files: `.project, build.xml and <bpm>.process file.`

15.2.4 Property Files

This section sets the guidelines for deploying property file artifacts.

Manual Steps

Copy the designated property file(s) from `<wm_install_dir>/IntegrationServer/packages/<package_name>/config` directory on the source server to the same directory on the target server. This should occur after the package has been installed.

Automated Steps

Step	Step Information
1	Create a <package_name>_Properties.properties file and enter the key/value pairs required. Note that this will overwrite the an existing properties file on the target server.
2	Make sure that CRLF is added after each line and after the last line. Also, make sure that special characters are escaped using two backslashes "\\" For example, http://www.yahoo.com properly escaped is: http:\\/\\/www.yahoo.com\\/
3	Check-in the property file into the designated source control system to `<package_name>/config` directory.

15.2.5 External Dependencies

This section sets the guidelines for deploying items outside of the deployment scope for webMethods but in cases where there is a dependency. Normally, external items require manual steps and are captured in detail within the deployment guide. The following outlines potential examples:

- Database object creation or modification. This may include setting up users or groups, setting permissions or creating/modifying database objects (like tables, stored procedures, views, etc.)
- PKI configuration setup
- X.509 Certificate setup
- EFD Configuration setup
- Centrasite/Mediator setup

16.0 AUTOMATED DEPLOYMENT WITH JENKINS

16.1 INTRODUCTION TO JENKINS CONTINUOUS INTEGRATION WITH WEBMETHODS

Automated deployment is accomplished using the webMethods Asset Build Environment (ABE) and webMethods Deployer. Jenkins can be added to act as a scheduling and orchestration tool to control webMethods ABE and webMethods Deployer execution steps.

webMethods ABE uses Ant to execute package build scripts supplied by Software AG. ABE is executed by calling a shell script which in-turn, executes the pre-packaged build.xml Ant scripts. Configuration properties with the ABE configuration files describe to ABE where to look for deployable webMethods components, and to where the resulting ABE generated artifacts should be staged.

Upon completion of the ABE script, the results are staged in a repository directory structure referred to as REPO. REPO becomes the source build from which webMethods Deployer deploys assets to target environments.

webMethods Deployer uses the following terms to describe its components:

Repository: A repository is an alias that defines the location on the file system where webMethods ABE has placed deployable assets that is has built. Repository definitions are independent of webMethods Deployer projects. A repository can be shared across multiple Deployer projects if desired.

Project: A project is a container for all other Deployer components such as the set, the map, and the candidate. Each are further defined below. Projects can either be runtime-based using a live webMethods instance for its source, or repository based, using an ABE built repository as its source.

Set: A deployment set is the set of assets that are to be included in the deployment. A project can have many deployment sets.

Build: A build is a snapshot of the assets that were selected within the set. In the case of a repository project, the build is the content of the repository when it was last indexed by an ABE build execution. For runtime projects, the project can have many builds which can later be defined in a deployment candidate and deployed.

Map: A deployment map, maps the project's deployment sets to one or more target servers.

Candidate: A deployment candidate is a combination of a build and a map. With repository projects, the build is represented by the contents of the repository the last time it was indexed by the ABE build execution.

For the NNASP Development region, only repository projects are being utilized. Projects are created on the fly using the Project Automator tool whereby the definition of the webMethods

Deployer project, set, map, and candidate are defined in the XML file. Project Automator uses the XML file to configure Deployer.

16.2 Installing Jenkins, webMethods Asset Build Environment (ABE), and Deployer

Jenkins is installed on the CI server under the following directory:

`/var/lib/Jenkins` (application dir)
`/etc./sysconfig/Jenkins` (configuration file)
Use `sudo /sbin/service Jenkins <start | stop | restart>` to manage the server

webMethods ABE is installed on the CI server under the following directory:

`/opt/ngafis/Interface/common/AssetBuiltEnvironment (ABE_HOME)`

webMethods Deployer is installed as an Integration Server Package under the following directory:

`/opt/ngafis/Interface/IntegrationServer/packages/WmDeployer (DEPLOYER_HOME)`

The following modifications were required for the NNASP Environment:

- Updated jar from SoftwareAG (ABE_DEBUG_FIX_IS_WmART.jar) was place in:

`/opt/ngafis/Interface/common/AssetBuildEnvironment/IS/lib/updates`

- Modified `<ABE_HOME>/bin/build.sh`:
  - Comment out the existing `JAVA_HOME`
  - Add new `JAVA_HOME=/usr/java/default`
  - Add export `ANT_HOME`
  - Increase `JAVA_MAX_MEM=1024M`
- Modified `<DEPLOYER_HOME>/bin/Deployer.sh`:
  - Comment out the existing `JAVA_HOME`
  - Add new `JAVA_HOME=/usr/java/default`
- Modified `<DEPLOYER_HOME>/bin/projectautomatorUnix.sh`:
  - Comment out the existing `JAVA_HOME`
  - Add new `JAVA_HOME=/usr/java/default`
- Added `<DEPLOYER_HOME>/config/NNASP_DeployerConfig.cnf` with:
  - `Host=<host or ip of webMethods Deployer Server>`
  - `port=<port for the webMethods Deployer Server>`
  - `user=<user to be used to execute deployments>`
  - `pwd=<password (clear text) for the user>`

In addition to the above changes, project automator XML files have been created and checked into SVN for each of the Jenkins defined projects. There are samples of files that get installed in the `<DEPLOYER_HOME/config>` directory. These files define how the command line interface (CLI) builds the webMethods Deployer project. Please refer to the Software AG documentation and the samples for further information.

16.3 OS Level Configuration for Interoperability between Jenkins and webMethods

The following OS level changes include:

- User `jenkins` was added to the `soa` group
- User `wmuser` was added to the `jenkins` group
- Users need to belong to the `jenkins` group in order to manage the server using `/sbin` commands

From `/opt/ngafis/Interface` directory, the following permissions were granted to the `soa` group:

- Directories: `find . -type d -exec chmod g+rx{}\;`
- Files: `find . -type f -exec chmod g+r{}\;`

16.4 Configuring Jenkins

Under Jenkins > Manage Jenkins > Configure System

Added global properties (environment variables):

- `ABE_BIN=/opt/ngafis/Interface/common/AssetBuildEnvironment/bin`
- `DEPLOYER_BIN=/opt/ngafis/Interface/IntegrationServer/packages/WmDeployer/bin`
- `DEPLOYER_CONFIG=/opt/ngafis/Interface/IntegrationServer/packages/WmDeployer/config`

16.5 Configuring Jenkins Jobs

Configuration of the interface specific jobs follows a fairly consistent pattern; therefore, creating a new job from an existing (working) job is highly recommended.

- Source code management is configured using as many repository locations as needed to perform the build using a check-out strategy of "Always check out a fresh copy".
- Build Triggers are set to poll SCM with a 15 minute interval scheduled as "0,15,30,45* * * *"
- The build steps are defined as a series of "Execute Shell" steps that perform necessary interactions with the various webMethods CLI commands.

  1. The first step is to remove `.svn` directories from the workspace using the following: `find ${WORKSPACE} -name *.svn -type d | xargs rm -rf`
  2. Next, update the project automator XML file that was checked out into the workspace to update the REPO location to be specific for each build. We decided to move `REPO` under the `builds` directory since the workspace is reused across multiple builds. We do not want

the REPO to span multiple builds, therefore placing it in the builds directory structure was the best fit. The following PERL script performs the update:

```
Perl -pi -e 's/##build/$ENV{BUILD_NUMBER}/g'
${WORKSPACE}/Deployer/${JOB_NAME}_ProjectAutomator.xml
```

3. Next, run the webMethods ABE script. This script uses the workspace as input, locates the assets that it recognizes as webMethods artifacts that it can deploy, and then outputs the necessary files into the REPO for those assets. The following is an example of running the build against the packages that were checked out from SVN into the Jenkins workspace. The webMethods ABE build script is executed once for each asset type being built, ie. IS, Broker, BPM, etc.

```
${ABE_BIN}/build.sh -Denable.source.dir=${WORKSPACE}/IS  -
Denable.output.dir=${WORKSPACE}/../builds/${BUILD_NUMBER}/REPO -
Denable.build.IS=true
```

4. Next, run the webMethods Deployer projectautomatorUnix.sh script. This step rebuilds the project inside of webMethods Deployer, defining such things as the target environments, the REPO location, and which assets from the REPO to deploy.

```
${DEPLOYER_BIN}/projectautomatorUnix.sh
${WORKSPACE}/Deployer/${JOB_NAME}_ProjectAutomator.xml
```

5. The final build step is to execute the Deployer script to actually deploy the assets. The following command is used:

```
${DEPLOYER_BIN}/Deployer.sh --deploy -configfile
${DEPLOYER_CONFIG}/NNASP_DeployerConfig.cnf -project
NNASP_${JOB_NAME} -dc ${JOB_NAME}Candidate -force
```

NOTES:

- Broker assets need to be named Broker_<interfaceName>.xml, in order to be picked up by the ABE build script as a Broker asset.
- BPM process model files must include build.xml in addition to the *.process file in order to be picked up by the ABE build script as a BPM asset.

17.0 Luke's Linux Tips and Tricks

Add in Luke's comments here regarding file permissions and manipulation.

APPENDIX A: BUILT-IN SERVICE REVIEW

This section provides an overview of select Flow/Java services that are packaged with the webMethods Integration Server. The intent is to draw attention to services whose purpose is not clear or where it may be used for several purposes beyond original intent. The format of the review is by folder and mirrors the Integration Server Built-In Services Guide.

ART Folder	
pub.art.transaction:startTransaction	Start a JDBC transaction
pub.art.transaction:commitTransaction	Commit a JDBC transaction
pub.art.transaction:rollbackTransaction	Rollback JDBC transaction
Comments: Used with transaction based JDBC calls (or any adapter under ART) to commit or rollback inserts, updates, or deletes. The JDBC Adapter configuration must be configured to use LOCAL_TRANSACTION. If the JDBC Adapter is configured to use NO_TRANSACTION, then the connection automatically commits operations. For multi-table operations, LOCAL_TRANSACTION will permit rollbacks across all operations.	

Cache Folder
Comments: Not included in the scope of the document. Built-in services to leverage Terracota. Folder present if Terracota licensed.

Client Folder	
pub. client.soapClient	Creates and sends SOAP 1.1 and SOAP 1.2 messages over HTTP/S, or JMS transports.
Comments: Client folder provides robust built-in methods for FTP, HTTP, LDAP, SMTP and SOAP. Most of the services are self-explanatory. NOTE: The status code may be extracted from the output of each call. For example, an HTTP status of 200 = OK but an HTTP status of 404 = Not Found.	

Date Folder	
pub.date:calculateDateDifference	Input 2 dates and determine difference in days, hours, minutes or seconds.
pub.date:dateTimeBuild	Builds a date/time string using the specified input pattern.
pub.date:getCurrentDate	Returns current date as date object. Useful for Java-based date calculations.
pub.date:getCurrentDateString	Returns current date as a string.
Comments: Provides useful date conversions and date builder features. One of the very confusing aspects of using the date services is the Pattern String Symbols required to return the desired data in the correct format.	

Db Folder
Comments: Folder contains supported Legacy services for the WmDB package. The WmDB package was the precursor to the JDBC Adapter. If a developer is having problems with the JDBC adapter, then using services in WmDB may be helpful for debugging purposes. The services in WmDB are not recommended for high-volume production environments. See section 13 for more information.

Document Folder	
pub.document:documentToXMLValues	Creates an XML String that represents a given pipeline document object. Used in this guide to determine how to build the quick document creator.
pub.document:XMLValuesToDocument	Creates a pipeline document object based on xmlvalues input.
Comments: Folder contains services to manipulate, sort, group or modify pipeline documents.	

Event Folder
Comments: Not included in the scope of the document. Built-in services to leverage events. These include exceptions, guaranteed delivery, JMS, Broker, Replication, Session and Security.

File Folder	
pub.file:bytesToFile	Writes a byte array to a file.
pub.file:checkFileExistence	Used to determine if a file is present prior to processing.
pub.file:getFile	**The most used service.** Obtains file contents as a string, stream or byte array.
pub.file:readerToFile	Writes a reader object to a file.
pub.file:streamToFile	Writes a stream object to a file.
pub.file:stringToFile	Writes a string object to a file.
Comments: Folder contains services to manipulate files and perform basic file system access and file I/O. NOTE: To write files, the ACL must be updated for the machine user to grant write permissions for the target directory. See the Administrator guide for more information.	

Flat File Folder
Comments: Not included in the scope of the document. Built-in services to leverage WmFlatFile built-in services. Not present if WmFlatFile is not installed.

Flow Folder	
pub.flow:clearPipeline	Used to clear the pipeline. Allows preservation of specified pipeline variables. Add variables into string array *preserve.*
pub.flow:debugLog	Use to write information or comments to the server log
pub.flow:restorePipeline	Restores pipeline from memory and aids in quicker testing because the data can be recalled without the need for input.
pub.flow:restorePipelineFromFile	Restores pipeline from a file. Allows developers to rerun services without needing to input data. It can be used to transport the pipeline to other IS servers or share the data with testers.
pub.flow:savePipeline	Saves pipeline to memory and aids in quicker testing because the data can be recalled without the need for input. Information not available after server reboot.
pub.flow:savePipelineToFile	Saves pipeline information into a file and should be used for persistent storage. Information is available after server reboot. Use in conjunction with restorePipelineFromFile.
Comments: **THE MOST USED FOLDER.**	

Hashtable Folder
Comments: Provides developer with a method to store variables during processing operations. The folder contains services that can create, update or obtain information. Uses key/value operations.

IO Folder
Comments: One of the most commonly used built-in flow services for data input/output. See advanced REPEAT flow code in section 8 for more information.

JMS Folder
Comments: Not included in the scope of the document. Built-in services to leverage the SAG JMS implementation on top of Broker.

List Folder
Comments: New services to work with Vector objects.

Math Folder	
pub.math:absoluteValue	Returns absolute value of the input number.
pub.math:randomDouble	Returns a pseudo random number between 0.0 and 1.0.
pub.math:randomNumber	Returns a rounded number
pub.math:toNumber	String to number
pub.math:addObjects pub.math:divideObjects pub.math:multiplyObjects pub.math:subtractObjects	Takes string input, castes to numeric object, performs calculations, recasts back to string and returns.
Comments: Contains documentation for the newest services.	

MIME Folder
Comments: Not included in the scope of the document. Used to compose or decompose email MIME objects. Used in conjunction with a configured email port.

Package Folder
Comments: Not included in the scope of the document. Contains utilities to manage packages

PKI Folder
Comments: Contains information related to certificate management. Not included in the scope of this document.

Publish Folder
Comments: Contains information related to Broker/Messaging. Not included in the scope of this document.

Remote Folder
Comments: Contains services required to remotely invoke flow services on a remote Integration Server. Manages guaranteed delivery (which specifies once and once only operations against a given input or message.

Replicator Folder
Comments: Not included in the scope of this document.

Report Folder
Comments: Not included in the scope of this document.

Scheduler Folder
Comments: Not included in the scope of this document. Manages IS repeating tasks.

Schema Folder
Comments: Not included in the scope of this document. Manages w3c schemas.

Security Folder
Comments: Not included in the scope of this document. Certificate management and cryptography classes.

SMIME Folder
Comments: Not included in the scope of this document. Manages secure email messages.

SOAP Folder
Comments: Not included in the scope of this document. Manages SOAP message composition and decomposition.

Storage Folder
Comments: Not included in the scope of this document. Original key/value storage mechanism. Largely replaced by the Hashtable folder. Stores persistent data into repository objects and can obtain key/values even if the server is restarted.

String Folder
Comments: Provides most common services required for String manipulation.

String Folder
Comments: Not included in the scope of this document.

Sync Folder
Comments: Not included in the scope of this document.

Synchronization Folder
Comments: Not included in the scope of this document.

Trigger Folder
Comments: Not included in the scope of this document.

UniversalName Folder
Comments: Not included in the scope of this document.

Utils Folder
Comments: Not included in the scope of this document.

VCS Folder
Comments: Not included in the scope of this document.

XML Folder
Comments: Not included in the scope of this document.

XSLT Folder
Comments: Not included in the scope of this document.

APPENDIX B: LINUX QUICK REFERENCE

Type "man + command" for help and a list of options/switches for a given command.

Linux Command	Description
File/Directory Basics	
cd	Change directory
cp	Copy files
ln	Link files
ls	List files
mkdir	Create directory
mv	Move files (or rename files)
pwd	Print current directory name
rm	Delete files
rmdir	Remove directory
File Viewing	
cat	View files
gv	View Postscript/PDF files
head	View file beginning
less	Page through files
nl	Number lines
od	View binary data
tail	View file ending
xdvi	View Tex DVI Files
xxd	View binary data
File Creation and Editing	
abiword	Edit Word documents
emacs	Text editor
gnumeric	Edit Excel documents
soffice	Edit Word/Excel/PowerPoint docs
umask	Set default file protections
vim	Text editor
File Properties	
chattr	Change advanced file attributes
chgrp	Change file group
chmod	Change file protections
chown	Change file owner
du	Measure disk usage
file	Identify file types
lattr	List advanced file attributes
stat	Display file attributes
touch	Change file timestamps
wc	Count bytes/words/lines
File Location	
find	Locate files
slocate	Locate files via index
whereis	Locate standard files
which	Locate commands

Linux Command	Description
File Text Manipulation	
`cut`	Extract columns
`grep`	Search text for matching lines
`paste`	Append columns
`sort`	Sort lines
`tee`	Copy stdin to a file and to stdout simultaneously
`tr`	Translate characters
`uniq`	Locate identical lines
File Comparison	
`comm`	Compare sorted files
`cmp`	Compare files byte by byte
`diff -u`	Compare files line by line
`md5sum`	Compute checksums
File Compression	
`bzip2`	Compress files (BZip2)
`compress`	Compress files (Unix)
`gzip`	Compress files (GNU zip)
`zip`	Compress files (Windows zip)
Disks and File Systems	
`df`	Show free disk space
`fsck`	Make a disk accessible
`mount`	Check a disk for errors
`sync`	Flush disk caches
Backups and Remote Storage	
`cdrecord`	Burn a CD
`dump`	Back up a disk
`mt`	Control a tape drive
`restore`	Restore a dump
`rsync`	Mirror a set of files
`tar`	Read/write tape archives

APPENDIX C: SUBVERSION QUICK REFERENCE

The following table represents a list of the most useful subversion commands:

Subversion Command	Description
SVN **--help**	List Subversion commands
SVN help *command* Also: ? or h	Help on given "*command*"
SVN **add** *filename* SVN add *directory*	Add a file or directory to Subversion CM control. Must also perform: SVN ci *filename* (or SVN commit) to upload the file or directory. File will not be available in the repository until a "commit" is performed. If adding a directory, the directory and all of its contents recursively are added. i.e.: SVN ci *directory* SVN commit *directory* SVN commit .
SVN **blame** *filename* SVN blame -r *RevisionNumber filename* Also: praise, annotate, ann	Show file contents with revisions annotated with author information.
SVN **cat** *filename*	List contents of file under Subversion control.
SVN **checkout** http://*node-name*/repos/svn/trunk/*parentPath/path* This creates: path/file1 path/file2 ... SVN **checkout** http://*node-name*/repos/svn/trunk/*parentPath* . This creates: path/file1 path/file2 ... Note the difference a "." makes. SVN checkout file:///repos/svn/trunk/*path*/ SVN co -r 497 http://*node-name*/repos/svn/trunk/*path file-name* Also: SVN co https://..., svn://..., and svn+ssh:// MS/Windows: SVN co file:///c:/repository/project/trunk	Checkout every file from the path and subdirectories specified below. Creates "working" copy of files and directories. Checkout a repository. Use option "-r" to specify a specific revision other than the latest. The URL "svn://" communicates with an SVN server (port 3690) The URL "http://" comunicates with the Apache server and module mod_dav_svn(port 80) [more common server]
SVN **cleanup**	Cleanup subversion files resulting from escaped

Subversion Command	Description
	processes and crashed.
SVN **commit** *filename* SVN commit --message "Message goes here." *filename* SVN commit -m "Message goes here." *filename* SVN ci *filename1 filename2 filename3* SVN ci .	Check-in (commit) local "working" file, files or directory and contents (recursively) into Subversion repository. Atomic, i.e. all committed or none, no incomplete check-in.
SVN **copy** *source destination_clone* Also: SVN cp ...	Copy file or directory tree. One can copy from one local working copy to another or to repository server URL's. The sources and destinations can be working copies or URLs.
SVN copy http://*host*/repos/project/trunk http://*host*/repos/project/**tags**/*TagName-1.4.5* -m "Tag Release 1.4.5"	Tag a release. Takes a snapshot of the repository and assigns a name. This can be performed at any directory branch.
SVN copy . http://*host*/repos/project/**tags**/*TagName-1.4.5* -m "Tag Release 1.4.5"	Tag a release. Takes a snapshot of your local working copy and assigns a name. This can be performed at any directory branch.
SVN **delete** *filename* SVN delete *directory* Also: del, remove or rm SVN rm http://host/repos/project/trunk/*file-or-directory*	Delete file from repository. The UNIX command rm *file-name*. Must perform a "commit" to update the repository and local working directory with the changes. i.e.: SVN commit .
SVN **diff** *filename* SVN di *filename*	Show file diffs between SVN repository and your file changes using GNU file diff format. Use GUI diff tools as shown below.
SVN diff -r *rev1:rev2 filename*	Show file diffs between specified versions. Example: SVN diff -r 456:459 *subfn.cpp* Using GUI diff tool: SVN diff -r 457:459 --diff-cmd kdiff3 *file-name*
SVN diff *filename* > *patch-file*	Generate patch file used by the patch command.

Subversion Command	Description
SVN **export** *directory*	Export directory tree to your file system but it will not be a "working directory" under SVN control.
SVN **export** -r *Rev-Number* http://*node-name/path*	Export directory tree of specified version and create local directory tree and files not under SVN control.
SVN **import** *local-directory* http://*node/repos/svn/trunk/directory*	Add directory (and files in it recursively) to path in repository specified.
SVN **info** *filename*	Display information about file or directory. (Date modified, author, revision, path in repository.) Can not specify a URL.
SVN **list** *directory* SVN list *file-name*	List file or directory of files in repository. Used to browse repository before checkout. If current directory is given (SVN list ./), then Subversion will list the repository URL of the current directory.
SVN list -r *RevisionNumber directory*	List directory of files in repository in specified revision.
SVN **lock** *filename* -m "*comment as to why its locked or by whom*" (Comment is not required but is often useful)	Lock file to grant exclusive access to one and forbid all others. A commit will unlock the file (unless the "--no-unlock" option is used). A lock can be removed with the commands: SVN unlock *filename*, svnlook and the svnadmin commands (i.e. List: svnadmin lslocks and remove: svnadmin rmlocks *filename*).
SVN **log** *filename* SVN log . SVN log http://*URL/path/file* SVN log -v . SVN log -r *RevisionNumber* http://*URL/path/file*	Show the Subversion log messages for a set of revision(s) and/or file(s) and/or all directory contents in repository.

Subversion Command	Description
	List verbose. Includes list of all files in change Shows the file changes associated with revision number.
SVN **merge** http://*url/path/branch1* http://*url/path/branch2* *working-local-dir* SVN merge *file1@revJ file2@revK* SVN merge -r 414:411 http://*url/path working-dir* SVN merge -r 413:HEAD *file-name*	Merge directory changes into your current working directory or merge a file in Subversion into the file in your working directory. If target is not specified, the identical basename or current directory is assumed. Used to incorporate changes checked in which are not accounted for in your file or to merge branches. Example using GUI merge tool: SVN diff -r 459:454 --diff-cmd kdiff3 --extensions '-m' *file-name* Next, tell subversion that the conflicts have been resolved: SVN resolve *file-name* Finally, check-in file: SVN ci *file-name* or abort changes: SVN revert *file-name*
SVN merge --dry-run -r 414:413 http://*url/path*	Test merge. No changes are made to your local working copy but shows Subversion feedback as if merge was performed.
SVN merge -r 414:413 http://*url/path* SVN merge -r 414:413 .	Undo changes committed in revision 414.
SVN **mkdir** *directory* SVN mkdir http://*URL/directory*	Create a new directory under version control.
SVN **move** *directory1 directory2* SVN mv *directory1 directory2* SVN mv *file-old-name file-new-name*	Rename or move a file or directory. Moves/renames file/directory in repository and in local work area. Must perform SVN ci *file-new-name* after the move for changes to to

Subversion Command	Description
	take place in repository.
SVN **revert** *filename*	Undo changes in local work files. Throw away local changes.
SVN **resolved** *filename*	Run this command after resolving merge conflicts. Next "`commit`" your changes.
SVN **status** SVN status -u SVN status -u . SVN status -uq .	Show status of file changes in current directory and recursively in directories below. Show out of date file info: SVN status --show-updates (equivalent: SVN status -u) -u: Determines status by comparing your local repository with the server repository. Without this option, the status shown will only be the changes you have made in your local repository. -q: Quiet. Do not print "?: File/directory not under version control" or "!: File/directory missing" extraneous information. First column: • A: File to be added • C: Conflicting changes • D: File to be deleted • G: File to be merged with updates from server • M: File has been modified • R: File to be replaced • G: File to be merged • X: Resource is external to repository (SVN:externals) • ?: File/directory not under version control • !: File/directory missing • ~: Versioned item obstructed by some item of a different kind.

Subversion Command	Description
	Second column: Modification of properties • ' ' no modifications. Working copy is up to date. • 'C' Conflicted • 'M' Modified • '*' Local file different than repository. A newer revision exists on the server. Update will result in merge or possible conflict. Third column: Locks • ' ' not locked • 'L' locked • 'S' switched to a branch
SVN **switch** http://server/new-branch SVN switch --relocate *http://server/old-path http://server/new-path*	Switch your local working copy to mirror a new repository branch instead of main trunk or previous branch. Also allows you to point your repository to a new path on the server if the server path changes since you performed a check-out.
SVN **update** SVN update *filename* SVN update -r458 *filename* SVN update --ignore-externals ./	Migrate all updates from Subversion repository to your local copy (recursively for all files in the current directory and all below it). If there have been updates to the SVN repository since you downloaded the files, subversion will give you the opportunity to merge. Status of files will use the coding as stated above for "status". Files marked with a "C" (conflict) should be merged of reverted. If merged then one can perform a "resolve" and then a "check-in". If a file name is specified,

Subversion Command	Description
	only that file is updated. Can also synchronize to a specified revision given by -r. Use --ignore-externals to avoid the slow processing of externals to a potentially slow distant internet server.

APPENDIX D: GIT QUICK REFERENCE FOR COMMON COMMANDS

GIT Command	Description
Branch & Stash	
`git branch`	List your branches. A "*" will appear next to the current active branch
`git branch` *`branch`*	Crate new branch at current branch
`git checkout` *`branch`*	Switch working directory to branch and check it out to your working directory
`git checkout -b` *`branch`*	Create a branch and immediately switch to it
`git checkout -b` *`new other`*	Branch *new* from *other* and switch to it
`git log`	Show commit logs
`git merge` *`branch`*	Merge another branch into one and record the merge as a commit
`git stash`	Stash away the currently uncommitted modifications in your working directory temporarily
`git stash apply`	Re-apply the last stashed changes
Conflicts – Use ***add*** to mark files as resolved.	
`git diff [--base]`	
`git diff --ours`	
`git diff --theirs`	
`git log --merge`	
`gitk --merge`	
Create	
`git init`	From existing files
`git add .`	
`git clone ~old/ ~/new`	From existing repository
`git clone git://`	
`git clone ssh://`	
Publish	
`git commit -a`	Add changed files automatically
`git format-patch origin`	Create set of diffs
`git push` *`remote`*	Push to origin or *remote*
`git tag` *`foo`*	Mark current version
Revert	
`git reset --hard`	No undo, resets to last commit
`git revert` *`branch`*	
`git commit -a --amend`	Replaces previous commit
`git checkout` *`id file`*	
Set Up and Init - GIT configuration and repository initialization and cloning	
`git config` *`key value`*	Set a configuration value in this repository
`git config`	Set a configuration value globally for this user
`git init`	Initialize an existing directory as a GIT repository
`git clone` *`url`*	Clone a GIT repository from a URL
`git help` *`command`*	Get help on any GIT command
Tracking Files	
`git add` *`files`*	Add files
`git mv` *`old new`*	Moves files from old directory to new directory

GIT Command	Description
`git rm files`	Remove files
`git rm --cached files`	Remove cached files
Update	
`git fetch`	From def. upstream
`git fetch remote`	
`git pull`	Fetch & merge
`git am -3 patch.mbox`	
`git apply patch.diff`	
Useful Tools	
`git archive`	Create release tarball
`git bisect`	Binary release for defects
`git cherry-pick`	Take a single commit from elsewhere
`git fsck`	Check Tree
`git gc`	Compress metadata (performance)
`git rebase`	Forward-port local changes to remote branch
`git remote add url`	Register a new remote repository for this tree
`git stash`	Temporarily set aside changes
`git tag`	(there's more to it)
`gitk`	TK GUI for Git
View	
`git status`	
`git diff [oldID newID]`	
`git log [-p] [file/dir]`	
`git blame file`	
`git show id`	Meta data + diff
`git show id:file`	
`git branch`	Shows list, * = current
`git tag -l`	Shows list