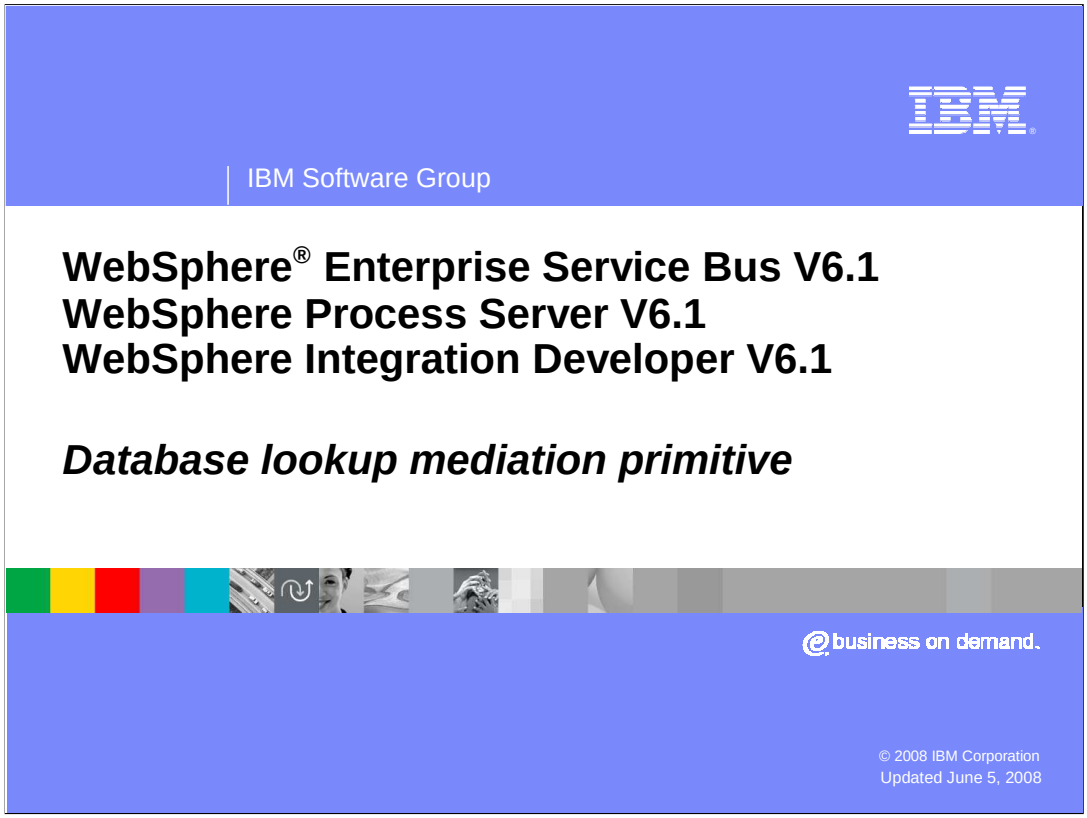


This presentation provides a detailed look at the Database lookup mediation primitive.

Goals

- Understand the database lookup primitive

Database lookup

  - Overview of function
  - Use of terminals
  - Definition of properties
  - Error handling
  - Example usage

The goal of this presentation is to provide you with a full understanding of the database lookup primitive.
The presentation assumes that you are already familiar with the material presented in the **Mediation Primitive Common Details** presentation and the **Common Details – Promoted Properties** presentation. These two presentations serve as a base for understanding mediation primitives in general.
In this presentation, an overview of the database lookup primitive is presented along with information about the primitive's use of terminals, its properties and error handling characteristics. Finally, an example usage of a database lookup is presented.

Overview of function

- Updates the Service Message Object (SMO) with data from a user database
  - A key value is obtained from the message
  - The database row associated with that key is accessed
  - The message is updated with values obtained from selected fields
- You must:
  - Create a database or use an existing database
  - Configure the data source identifying the database

The purpose of the database lookup is to update the Service Message Object (SMO) with data obtained from a user database.

The database lookup primitive first obtains a key value from the SMO and uses that key value to access a row from a database table. Values from selected fields in that row are then used to update the SMO. Depending upon your requirements, you must either create a new database or use an existing database. You must configure a data source that is used to identify the database.

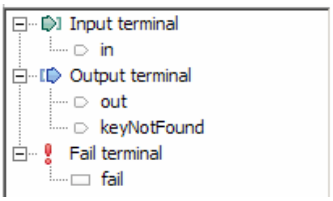
Overview of function

- Configuration data used to define database access
  - Data source JNDI name
  - Table name
  - Key column
  - Data columns
- Configuration information for the SMO:
  - XPath expression identifying the element containing the key value
  - XPath expressions identifying the message elements to update
  - Validation of input SMO

There is quite a bit of configuration data associated with a database lookup primitive. Some of the configuration data is associated with the database and other configuration information refers to the SMO. The database configuration information includes the JNDI name for the data source, the name of the table on which to do the lookup, the column used for the key and the columns from which to the extract data. The SMO configuration data includes the XPath expression that identifies the key value element in the SMO and the XPath expressions identifying the elements in the SMO to be updated with values from the database. There is also a configuration option to request validation of the incoming SMO.

Terminals

- Terminals:
  - Input terminal
  - Two output terminals
  - Fail terminal

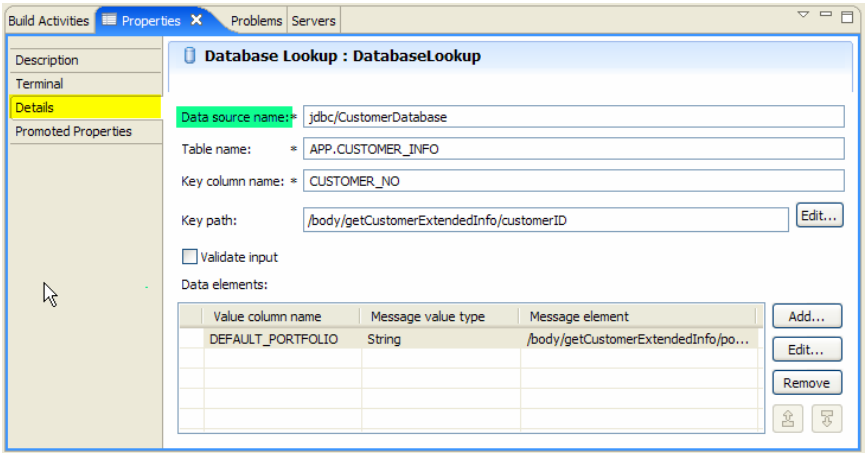


- Output terminals
  - **out** – Database record found, the updated message is propagated
  - **keyNotFound** – Database record not found, the unchanged message is propagated
- All terminals must be for the same message type

The database lookup primitive has one input terminal, two output terminals and a fail terminal. There is one output terminal named **out** that is used when the database lookup is successful. The other output terminal, named **keyNotFound,** is used when there is no row in the database for the key value obtained from the SMO. The output terminals must be for the same message type as the input terminal as the database lookup primitive does not modify the message body structure, although it does update the message body contents. The slide shows a database lookup primitive with its terminals and the terminals as seen in the properties view.

Properties

- Data source name
  - The JNDI name of the data source defining the database

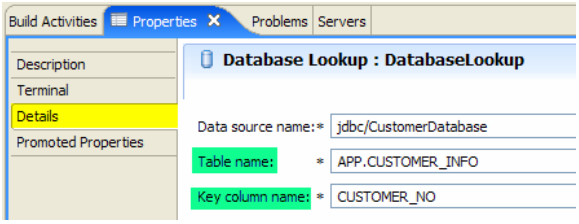


This screen capture shows the Details tab from the Properties view for a database lookup. The next several slides examine the various properties used to configure this primitive.

The **Data source name** property is a JNDI name used to lookup the data source that identifies the database containing the table to be used for the lookup. When creating a new database lookup primitive you must set this property as it does not have a default value.

Properties (continue)

- Table name
  - The fully qualified database table name on which to do the lookup
    - **schema.tableName**
- Key column name
  - The name of the primary key column to use for the lookup
  - Value must be unique
  - Multi-column keys are not supported

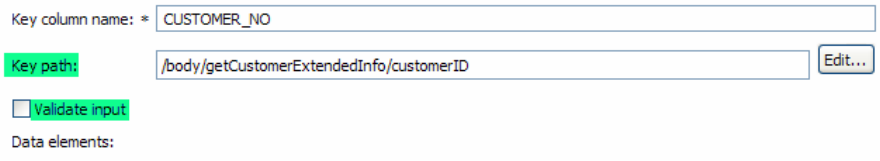


The **Table name** property contains the fully qualified database table name on which to do the lookup. In some cases using just the unqualified table name works. However, you should use the fully qualified schema.tableName value for this property as that always yields the expected results.

The **Key column name** property identifies the name of the primary key column to be used for the lookup. The key value must be unique within the database table because the database lookup primitive only works with a single row. Also, the use of multi-column keys is not supported.

Properties (continue)

- Key path
  - XPath expression
  - Identifies the message element containing the key value
- Validate input
  - Validates if incoming message is of the expected type
  - Ensures it meets any constraints defined
    - For example, minOccurs, maxValue, and so on

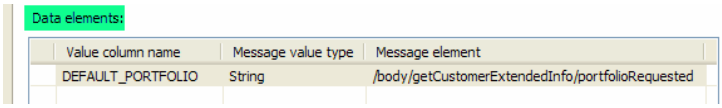


The **Key path** property contains an XPath expression which identifies the element within the SMO whose value is used as the key for the lookup. This expression must identify a single element in the SMO. The **Edit…** button can be used to access the XPath expression builder dialog to construct the XPath expression to be used.

The **Validate input** property is a check box used to indicate if incoming messages to the database lookup primitive are to be validated before processing. This ensures that the incoming message is of the expected type and that any constraints defined are not violated.

Properties (continue)

- Data elements
  - Table with list of elements to be updated



  - Each data element contains:
    - **Value column name** – name of the database column containing the source value
    - **Message value type** – data type of element to be updated in message
      - Must be a String or Java™ primitive type
      - Source value is converted to this type if needed
    - **Message element** – XPath expression identifying element to be updated
      - Must be a single element
      - If value already exists, it is overridden

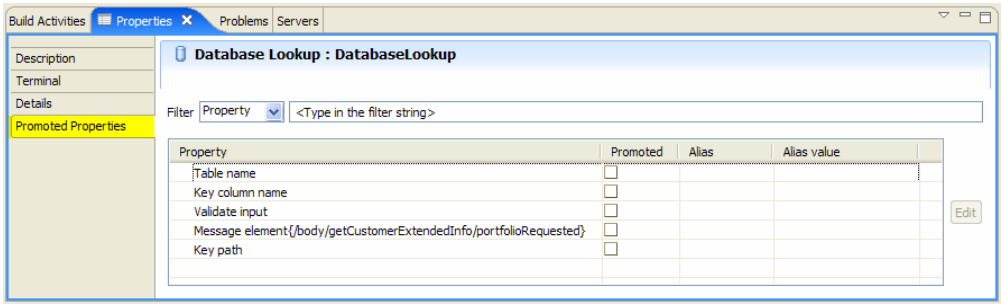
The final property for the database lookup primitive is the **Data elements** property. It is a table base property with three columns. Each row in the table identifies a specific element within the SMO that is updated using data obtained from the database.

The first column in the table is called the **Value column name** and it identifies the column in the database from which the source value for the update is obtained.

The next column is called the **Message value type** and it defines the data type within the SMO for the element to be updated. This must be either a String or a primitive Java type. The source value obtained from the database column is converted to this type before being inserted into the SMO. If the value can not be converted to the specified type an exception is raised.

The third column is the **Message element** which identifies the element in the SMO to be updated. This is specified with an XPath expression and must identify a single element within the SMO. If a value for that element already exists in the SMO, it is overwritten.

Promoted properties



- Promotable
  - Table name
  - Key column name
  - Key path
  - Validate input
  - Data elements (Message element column)
- Not promotable
  - Data source name

This slide shows the Promoted Properties panel and lists those properties that are promotable and those that are not.

There are four properties which play some part in identifying aspects of the database being used for the lookup, and only two of those properties are promotable. Starting from the highest level of abstraction is the data source name, which is a JNDI name identifying the database to use. This is not a promotable property because it is used as part of a resource reference with a generated EJB, and changing this property at runtime requires a redeployment of the mediation application. The next two, the table name and key column name are promotable. Finally, within the data elements table, there is a value column name which is not promotable. This is not promotable because a different column in the table was designated as promotable and only one column in a table based property can be designated as promotable.

The remaining properties are related to the SMO. The key path property and validate input properties are promotable. Also, within the data elements table based property the message element column is promotable.

Error processing

- MediationRuntimeException thrown for:
  - Incorrect JNDI name for data source
  - Key element is not found in the SMO
- MediationBusinessException (fail terminal flow)
  - Key element in SMO contains a null value
  - Validate input specified and message fails validation testing
  - Unable to convert value from database to specified message value type
  - Message element XPath does not identify a single element in message
- MediationConfigurationException (fail terminal flow)
  - Incorrect name for database table, key column or value column
  - Cannot connect to database

The error processing details and considerations are examined on this slide.

A MediationRuntimeException is thrown for an incorrect JNDI name for the data source. It is also thrown if the key element does not exist in the SMO, indicating a problem with the configured XPath expression for the key element.

A MediationBusinessException occurs for several different problems.

One cause is that the element for the key value in the SMO contains a null value. This is different than the previous situation that causes a MediationRuntimeException in that the element actually does exist in this case. The problem here is that the element has a null value and therefore cannot be used as a key in the lookup.

Another cause of the MediationBusinessException is when the validate input property has been specified and the message fails the validation processing.

The MediationBusinessException also occurs when the value taken from the database cannot be converted to the specified message value type. For example, if the message type was “float” and the value obtained from the database was “abc”.

Another possible reason is if the XPath expression for the message element identified something in the SMO which was not a single element. In all of these cases, if the fail terminal is wired, an exception is not thrown and the flow continues, following the wire from the fail terminal.

A MediationConfigurationException occurs for any kind of issues with the configuration of the database, such as an incorrect table name, key column name or value column name. It also occurs if there are any other problems accessing the database. In these cases, if the fail terminal is wired the flow from the fail terminal is followed rather than the exception being thrown.

Example usage

- Example – Set a value for a missing value in request
  - Customers can have multiple portfolios
  - Account number and portfolio ID are contained in request
  - Some requests might not have the portfolio ID specified
  - Need to initialize portfolio ID to a customer specific default value
- Mediation logic
  - A message filter checks if the portfolio ID is null
  - When portfolio ID is null, the database lookup is used
  - Database lookups must be successful to proceed

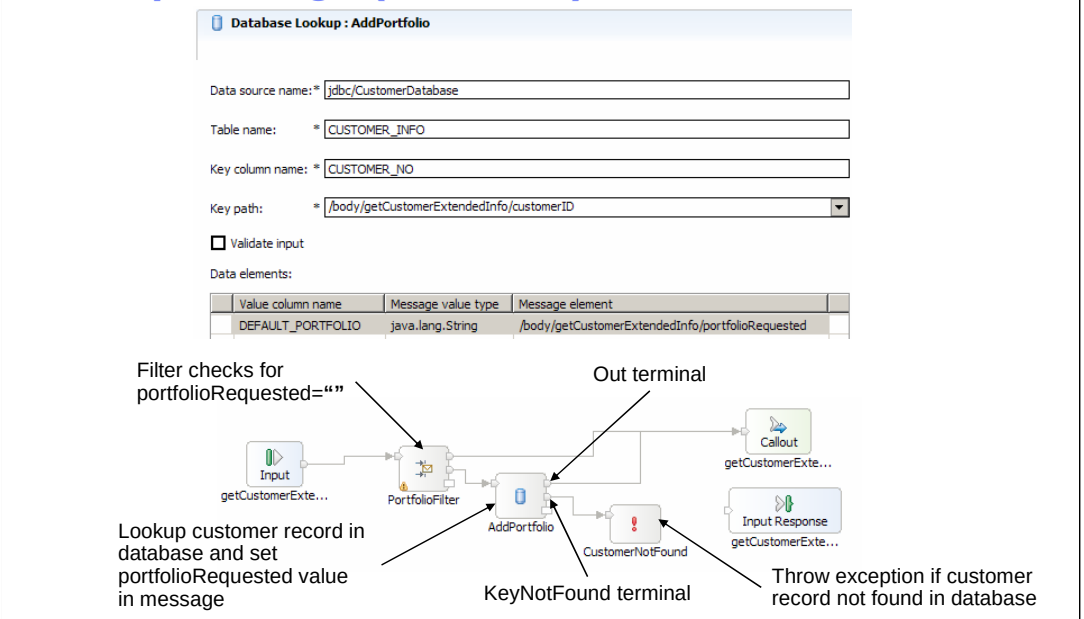
In the next couple of slides you are shown an example usage of the database lookup primitive. The example looks at a case where some service requests do not contain all of the data required by the service provider. Therefore, the database lookup is used to obtain the data from a database and insert it into the message.

In this scenario, customers can have multiple portfolios. The requests to the service provider should contain the customer account number and the portfolio ID. However, some service requests only contain the customer account number. The database lookup is used to determine the default portfolio ID for that customer's account and places it into the message.

The logic in the mediation flow starts with a message filter that is used to check if the portfolio ID in the message is null. When the portfolio ID is null, a database lookup is used to obtain the default portfolio ID for that specific customer. In the event that the database lookup fails, it is considered an error.

The next slide shows the database lookup properties and the mediation flow used in this scenario.

Example usage (continue)



The top part of the slide shows the properties for the database lookup. The lookup is done on the CUSTOMER_INFO table with a key column of CUSTOMER_NO. The key value is obtained from the customerID element of the SMO message body. In the data elements table you can see that the value in the DEFAULT_PORTFOLIO column is obtained and used to update the portfolioRequested element in the SMO body.

The bottom of the slide shows the mediation flow. The message is first passed to the PortfolioFilter message filter primitive which determines if the portfolio ID is contained in the request. If it is, the flow goes directly to the callout for the service provider. If not, the AddPortfolio database lookup primitive is used to access the customer information in the database and update the SMO with the portfolio ID. If the lookup is successful the flow goes to the callout for the service provider. If the lookup is not successful, the keyNotFound terminal is wired to a fail primitive which causes an exception to be raised.

Summary

- Examined the database lookup primitive

Database lookup

  - Overview of function
  - Use of terminals
  - Definition of properties
  - Error handling
  - Example usage

This presentation covered details about the database lookup primitive. The presentation covered an overview of the database lookup along with information about the primitive's use of terminals, its properties and error handling characteristics. Finally, an example usage of a database lookup was presented.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WBIV61_DatabaseLookupPrimitive.ppt

This module is also available in PDF format at: ../WBIV61_DatabaseLookupPrimitive.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

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

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.