312
JMS Types

PL/SQL users can use the `DBMS_AQ` package to enqueue and dequeue messages from JMS queues.

The JMS types member and static functions and procedures in this chapter are needed to populate JMS messages for enqueuing or to interpret a dequeued JMS message.

This chapter contains these topics:

- Overview
- Security Model
- Java Versus PL/SQL Datatypes
- More on Bytes_ Stream and Map Messages
- Upcasting and Downcasting Between General and Specific Messages
- JMS Types Error Reporting
- Oracle JMS Type Constants
- JMS Types Error Reporting
- Oracle JMS Type Constants
- CONVERT_JMS_SELECTOR
- Summary of JMS Types

JMS Types Overview

Java Message Service (JMS) is a well known public standard interface for accessing messaging systems. Oracle JMS (OJMS) implements JMS based on Oracle Advanced Queuing (AQ) and a relational database system (RDBMS). Messages are stored in queues as OJMS specific ADTs. Java clients use OJMS packages to enqueue, dequeue, and manipulate these messages.

PL/SQL users, on the other hand, use the `DBMS_AQ` package to enqueue and dequeue JMS messages and the member functions in this chapter to populate and interpret them. Oracle Database Advanced Queuing offers such member functions for the following JMS ADTs:

- `aq$_jms_header`
- `aq$_jms_message`
- `aq$_jms_text_message`
- `aq$_jms_bytes_message`
- `aq$_jms_map_message`
- `aq$_jms_stream_message`

In addition to these populating and interpreting member functions, Oracle Database Advanced Queuing offers:

- Casting between `aq$_jms_message` and other message ADTs.
- PL/SQL stored procedures for converting JMS selectors to equivalent Oracle Database Advanced Queuing rules

JMS Types Security Model

`PUBLIC` is granted `EXECUTE` privilege in these JMS types.

- SYS.AQ$_JMS_MESSAGE Type
- SYS.AQ$_JMS_TEXT_MESSAGE Type
- SYS.AQ$_JMS_BYTES_MESSAGE Type
- SYS.AQ$_JMS_MAP_MESSAGE Type
- SYS.AQ$_JMS_STREAM_MESSAGE Type
- SYS.AQ$_JMS_OBJECT_MESSAGE Type
- SYS.AQ$_JMS_NAMEARRAY Type
- SYS.AQ$_JMS_VALUE Type
- SYS.AQ$_JMS_EXCEPTION Type

Java Versus PL/SQL Datatypes

Datatypes do not map one-to-one between PL/SQL and Java.

Some Java types, such as `BYTE` and `SHORT`, are not present in PL/SQL. PL/SQL type `INT` was chosen to represent these types. If a PL/SQL `INT` value intended to hold a Java `BYTE` or `SHORT` value exceeds the corresponding range Java enforces, an out-of-range error is thrown.

Other Java types have more than one counterpart in PL/SQL with different capabilities. A Java String can be represented by both `VARCHAR2` and `CLOB`, but `VARCHAR2` has a maximum limit of 4000 bytes. When retrieving `TEXT` data from map, stream, and bytes message types, a `CLOB` is always returned. When updating the map, stream and bytes message types, users can submit either a `VARCHAR2` or `CLOB`.

Similarly, a Java `BYTE ARRAY` can be represented by both `RAW` and `BLOB`, with `RAW` having a maximum size of 32767. When retrieving `BYTE ARRAY` data from map, stream, and bytes message types, a `BLOB` is always returned. When updating the map, stream and bytes message types, users can submit either a `RAW` or `BLOB`.

See Also:

JMS specification 3.11.3, Conversion Provided by StreamMessage and MapMessage

New JMS Support in Oracle Database 10*g*

In Oracle Database 10*g*, a new `AQ$_JMS_VALUE` ADT has been added in the `SYS` schema for OJMS PL/SQL users. It is specifically used to implement the `read_object` procedure of `aq$_jms_stream_message` and `get_object` procedure of `aq$_jms_map_message`, to mimic the Java general object class `Object`. `AQ$_JMS_VALUE` ADT can represent any datatype that JMS `StreamMessage` and `MapMessage` can hold.

The collection ADT `AQ$_JMS_NAMEARRAY` was added for the `getNames` method of `MapMessage`. It holds an array of names.

In this release the ADT `AQ$_JMS_EXCEPTION` was added to represent a Java exception thrown in an OJMS JAVA stored procedure on the PL/SQL side. Now you can retrieve a Java exception thrown by an OJMS stored procedure and analyze it on the PL/SQL side.

More on Bytes, Stream and Map Messages

Oracle uses Java stored procedure to implement some of the procedures of `AQ$_MAP_MESSAGE`, `AQ$_JMS_STREAM_MESSAGE`, and `AQ$_JMS_BYTES_MESSAGE` types. These types have some common functionality that are different from `AQ$_JMS_TEXT_MESSAGE` type. This section discusses this common functionality.

This section contains these topics:

- Using Java Stored Procedures to Encode and Decode Oracle Database Advanced Queuing Messages
- Initialize the Jserv Static Variable
- Get the Payload Data Back to PL/SQL
- Garbage Collect the Static Variable
- Use a Message Store: A Static Variable Collection
- Typical Calling Sequences
- Read-Only and Write-Only Modes Enforced for Stream and Bytes Messages
- Differences Between Bytes and Stream Messages
- Getting and Setting Bytes, Map, and Stream Messages as RAW Bytes

Using Java Stored Procedures to Encode and Decode Oracle Database Advanced Queuing Messages

The major difference between map, stream, bytes, and other messages is that the message payload is encoded as a byte stream by JAVA. Retrieving and updating these payloads in PL/SQL therefore requires Oracle JAVA stored procedures.

A message payload is stored in two places during processing. On the PL/SQL side it is stored as the data members of a JMS message ADT, and on the Jserv side it is stored as a static variable. (Jserv is the JVM inside Oracle Database.) When the payload is processed, the payload data is first transformed to a static variable on the Jserv side. Once the static variable is initialized, all later updates on the message payload are performed on this static variable. At the end of processing, payload data is flushed back to the PL/SQL side.

Oracle provides member procedures that maintain the status of the Jserv static variable and enforce rules when calling these member procedures. These procedures are in the following ADTs:

- `aq$_jms_bytes_message`
- `aq$_jms_map_message`
- `aq$_jms_stream_message`

Initialize the Jserv Static Variable

Before you make any other calls to manipulate the payload data, the Jserv static variable must be properly initialized. This is done by calling the `prepare` or `clear_body` procedure. The `prepare` procedure uses the payload data in PL/SQL ADTs to initialize the static variable, while `clear_body` initializes the static variable to an empty payload (empty hashtable or stream).

> **Note:**
>
> It is important to call the `prepare` or `clear_body` procedure before any other calls to properly initialize the Jserv static variables. Usually these two methods are called once at the beginning. But they can be called multiple times for one message. Any call of these two methods without first calling the `flush` procedure wipes out all updates made to the messages.

Get the Payload Data Back to PL/SQL

Calling the `flush` procedure synchronizes changes made to the Jserv static variable back to the PL/SQL ADTs. The `flush` call is required when you want the changes made to be reflected in the ADT payload. It is important to synchronize the changes back to the ADT, because it is the ADT payload that matters.

Garbage Collect the Static Variable

The `clean` procedure forces garbage collection of the static variable. It is there to do cleanup and free JVM memory. You can avoid memory leaks by doing it immediately after finishing processing the message.

Use a Message Store: A Static Variable Collection

Instead of a single static variable, Oracle uses a collection of static variables to process the message payload on the Jserv side. This collection is called the message store. Each map, bytes, or stream message type has its own message store within one session.

Oracle uses the operation ID parameter to locate the correct static variable to work on within the message store. Initialization calls such as `prepare` and `clear_body` give users an operation ID, which is used in later message access.

After users complete message processing, they must call the `clean` procedure with the operation ID to clean up the message store. This avoids possible memory leaks. The `clean_all` static procedures of message ADTs `aq$_jms_bytes_message`, `aq$_jms_map_message`, and `aq$_jms_stream_message` clean up all static variables of their corresponding message stores.

Typical Calling Sequences

This section describes typical procedures for retrieving and populating messages.

Here is a typical procedure for retrieving messages

1. Call `prepare` for a message.

   This call also gives you an operation ID if you do not specify one.
2. Call multiple retrieving procedures with the provided operation ID.

3. Call the `clean` procedure with the provided operation ID.

Here is a typical procedure for populating messages:

1. Call `clear_body` for a message.

   For `aq$_jms_map_message`, you can also call `prepare` to update the message based on the existing payload. This call also gives you an operation ID if you do not specify one.
2. Call multiple updating procedures with the provided operation ID.
3. Call the `flush` method with the provided operation ID.
4. Call the `clean` procedure with the provided operation ID.

Read-Only and Write-Only Modes Enforced for Stream and Bytes Messages

According to the JMS specification, when a message is received, its body is read-only. Users can call the `clear_body` method to make the body writable. This method erases the current message body and sets the message body to be empty.

The OJMS JAVA API follows the rule set by JMS specification. In updating the JMS message ADTs in PL/SQL, however, Oracle enforces the rule selectively:

- Map messages

  The restriction is relaxed, because adding more entries on top of a existing map payload is a convenient way for users to update the payload. Therefore there are no read-only or write-only modes for map messages.
- Stream and bytes messages

  The restriction is not relaxed, because these payloads use a stream when reading and writing data. It is difficult to update the payload while in the middle of a stream. Oracle enforces read-only and write-only modes in processing stream and bytes message payloads. Calling the `prepare` procedure initializes the message payload in read-only mode. Calling the `clear_body` procedure initializes the message payload in write-only mode.

  Calling the `reset` procedure resets the pointer to the beginning of the stream and switches the mode from write-only to read-only. The `reset` procedure keeps the updates made to the message payload in the Jserv static variable.

  The `prepare` procedure, on the other hand, overwrites the message payload in the Jserv static variable with the payload in the PL/SQL ADT.

  Oracle provides member function `get_mode` for users to query the mode.

Differences Between Bytes and Stream Messages

Member functions of bytes messages are not exactly the same as those of stream messages. Stream messages are encoded using Java `ObjectOutputStream` and bytes messages are encoded using Java `DataOutputStream`. In stream messages each primitive type is written and read as a Java Object, but in a bytes message they are written and read as raw bytes according to the encoding mechanism of `DataOutputStream`.

For stream messages, the `read_bytes` method works on a stream of bytes to the end of the byte array field written by the corresponding `write_bytes` method. The `read_bytes` method of bytes message works on a stream of bytes to the end of the whole byte stream. This is why the `read_bytes` member procedure of `aq$_bytes_message` also requires a `length` parameter to tell how long it is to read.

You will not see a type conversion error raised by bytes message, because bytes messages do not support type conversion.

Methods `get_unsigned_byte` and `get_unsigned_short` are available for bytes messages, but not for stream messages. This is because stream messages read Java objects, and there are no Java objects as unsigned bytes or unsigned shorts.

Methods `read_string` and `write_string` methods are not available for bytes messages. The bytes message ADT must enforce some character encoding. It has methods `read_utf` and `write_utf` which support `utf-8` encoding.

Note:

All data written by bytes messages use `DataOutputStream` as the basis. See JDK API documentation JavaSoft.com for details on how the data is encoded into bytes.

Getting and Setting Bytes, Map, and Stream Messages as RAW Bytes

The payloads of bytes, map, and stream message types are stored as either `RAW` or `BLOB` in the database. In this release Oracle Database Advanced Queuing provides the following member functions to set and get these payloads as raw bytes without interpreting them:

```
set_bytes(payload IN BLOB)
set_bytes(payload IN RAW)
get_bytes(payload OUT BLOB)
get_bytes(payload OUT RAW)
```

These functions were provided for bytes messages in Oracle9*i* Release 2 (9.2).

Upcasting and Downcasting Between General and Specific Messages

OJMS ADT `aq$_jms_message` is used to represent a general message, so that different types of messages can reside on the same Oracle Database Advanced Queuing queue. Oracle Database Advanced Queuing supports retrieving and populating of `aq$_jms_message` by supporting upcasting and downcasting between this ADT and ADTs of specific message types.

To read an `aq$_jms_message`, you must first downcast it to a specific message type according to its `message_type` field

To populate an `aq$_jms_message`, you must first populate a specific message and upcast it to `aq$_jms_message`. This avoids copying all member functions of other specific message ADTs to this ADT. It also guarantees that the manipulation of this ADT is consistent with other specific message ADTs.

JMS Types Error Reporting

This table lists Oracle JMS types related errors.

Table 312-1 Oracle JMS Types Errors

ORA error number	dbms_jms_plsql package constants	Explanation
ORA-24190	`ERROR_DATA_OVERFLOW`	The payload data exceeds the size that an out parameter can hold. For example, the `get_text` procedure with a `VARCHAR2` parameter of `aq$_jms_text_message` or `get_bytes procedure` with a `RAW` parameter of `aq$_jms_bytes_message.`
ORA-24191	`ERROR_PROP_NAME_EXIST`	Setting a property that is previous set
ORA-24192	`ERROR_PROP_NAME_NULL`	Occurs when setting a property with null property name.
ORA-24193	`ERROR_EXCEED_RANGE`	PL/SQL number type exceeds the valid range of the respective Java type. For example `set_byte_property`, `set_short_property` of `aq$_jms_head` ADT; `set_byte` and `set_short` of `aq$_jms_map_message` ADT; `write_byte` and `write_short` of `aq$_jms_stream_message` and `aq$_jms_bytes_message` ADT.
ORA-24194	`ERROR_TYPE_MISMATCH`	The type conversion between the Java type of the retrieving method and the Java type of a field of the payload is not valid.
ORA-24195	`ERROR_MAP_TOO_LARGE`	The size of the map exceeds the `aq$_jms_namearray` ADT capacity. The current size limit is 1024. You can use the `get_names` function with `offset` and `length` parameters to retrieve the name array in multiple small chunks.
ORA-24196	`ERROR_WRONG_MODE`	The message payload is being accessed with a wrong access mode. For example, trying to read a message payload with write-only mode or trying to write a message payload with the read-only mode.
ORA-24197	`ERROR_JAVA_EXCEPTION`	ORA-24197 error is raised when a Java exception is raised that does not fit in any of the other error categories. You can use the `get_exception` static procedure of `aq$_jms_map_message`, `aq$_jms_bytes_message`, and `aq$_jms_stream_message` to retrieve the exception information last thrown by the Java stored procedure. A single static variable is used to store the last exception and is overwritten if another exception is thrown before you retrieve it. A new ADT `aq$_jms_exception` is created to represent the exception information on the PL/SQL side.
ORA-24198	`ERROR_INVALID_ID`	An invalid operation ID is being provided to access a message.
ORA-24199	`ERROR_STORE_OVERFLOW`	The number of messages (with the same type) that users are trying to manipulate exceeds the size of the message store on the Java stored procedure side. The current size of the store is 20. It unusual to need to manipulate more than 20 messages at the same time. A common mistake is to forget to call the `clean` procedure after using one message. The `clean` procedure frees the message slot for use by other messages attempting access.

Oracle JMS Type Constants

These constants can be useful when dealing with message type functions.

DBMS_AQ Package Constants

`DBMS_AQ` package constants specify different types of JMS messages. They are useful when dealing with general message types during upcasting and downcasting or constructing a general message with a specific message type:

```
JMS_TEXT_MESSAGE   CONSTANT BINARY_INTEGER;
JMS_BYTES_MESSAGE  CONSTANT BINARY_INTEGER;
JMS_STREAM_MESSAGE CONSTANT BINARY_INTEGER;
JMS_MAP_MESSAGE    CONSTANT BINARY_INTEGER;
JMS_OBJECT_MESSAGE CONSTANT BINARY_INTEGER;
```

SYS.DBMS_JMS_PLSQL Package Constants

`SYS.DBMS_JMS_PLSQL` package constants are new in Oracle Database 10*g*.

These constants specify the mode of message payload. They are useful when interpreting the mode of the message payload returned from the `get_mode` function:

```
MESSAGE_ACCESS_READONLY  CONSTANT PLS_INTEGER;
MESSAGE_ACCESS_WRITEONLY CONSTANT PLS_INTEGER;
```

These constants specify the ADT type of an Oracle Database Advanced Queuing queue. They are useful during the conversion of JMS selectors to Oracle Database Advanced Queuing rules:

```
DESTPLOAD_JMSTYPE CONSTANT PLS_INTEGER;
DESTPLOAD_USERADT CONSTANT PLS_INTEGER;
DESTPLOAD_ANYDATA CONSTANT PLS_INTEGER;
```

These constants specify the type of data that can be held by a `aq$_jms_value` type. They are useful when interpreting the `aq$_jms_value` returned by the `get_object` method of `AQ$_JMS_MAP_MESSAGE` or `read_object` method of `AQ$_JMS_STREAM_MESSAGE`:

```
DATA_TYPE_BYTE           CONSTANT PLS_INTEGER;
DATA_TYPE_SHORT          CONSTANT PLS_INTEGER;
DATA_TYPE_INTEGER        CONSTANT PLS_INTEGER;
DATA_TYPE_LONG           CONSTANT PLS_INTEGER;
DATA_TYPE_FLOAT          CONSTANT PLS_INTEGER;
DATA_TYPE_DOUBLE         CONSTANT PLS_INTEGER;
DATA_TYPE_BOOLEAN        CONSTANT PLS_INTEGER;
DATA_TYPE_CHARACTER      CONSTANT PLS_INTEGER;
DATA_TYPE_STRING         CONSTANT PLS_INTEGER;
DATA_TYPE_BYTES          CONSTANT PLS_INTEGER;
DATA_TYPE_UNSIGNED_BYTE  CONSTANT PLS_INTEGER;
DATA_TYPE_UNSIGNED_SHORT CONSTANT PLS_INTEGER;
```

These constants specify the error number of the ORA errors that can be raised by the functions of message type ADTs. They are useful in user error handlers:

```
ERROR_DATA_OVERFLOW   CONSTANT PLS_INTEGER := -24190;
ERROR_PROP_NAME_EXIST CONSTANT PLS_INTEGER := -24191;
ERROR_PROP_NAME_NULL  CONSTANT PLS_INTEGER := -24192;
ERROR_EXCEED_RANGE    CONSTANT PLS_INTEGER := -24193;
ERROR_TYPE_MISMATCH   CONSTANT PLS_INTEGER := -24194;
ERROR_MAP_TOO_LARGE   CONSTANT PLS_INTEGER := -24195;
ERROR_WRONG_MODE      CONSTANT PLS_INTEGER := -24196;
ERROR_JAVA_EXCEPTION  CONSTANT PLS_INTEGER := -24197;
ERROR_INVALID_ID      CONSTANT PLS_INTEGER := -24198;
ERROR_STORE_OVERFLOW  CONSTANT PLS_INTEGER := -24199;
```

CONVERT_JMS_SELECTOR

Oracle Database includes three stored procedures to help users convert JMS selectors into Oracle Database Advanced Queuing rules. These rules can be used in `ADD_SUBSCRIBER` operations as subscriber rules or in `DEQUEUE` operations as dequeue conditions. These procedures are in the `SYS.dbms_jms_plsql` package.

Convert with Minimal Specification

The first procedure assumes the destination payload type is one of the JMS ADTs whose corresponding constant is `dbms_jms_plsql.DESTPLOAD_JMSTYPE` and also assumes that the J2EE compliant mode is true.

Syntax

```
Function convert_jms_selector(selector IN VARCHAR2) RETURN VARCHAR2
```

Returns

The converted Oracle Database Advanced Queuing rule or null if there is any conversion error.

Exceptions

ORA-24197 if the Java stored procedure throws an exception during execution.

Convert with Destination Payload Type Specified

The second procedure takes one more parameter: `dest_pload_type`. The conversion of a JMS selector to an Oracle Database Advanced Queuing rule happens only if this parameter is `SYS.dbms_jms_plsql.DESTPLOAD_JMSTYPE` or `SYS.dbms_jms_plsql.DESTPLOAD_ANYDATA`. The function returns exactly the same `VARCHAR2` value as the selector parameter if the `dest_pload_type` parameter is `SYS.dbms_jms_plsql.DESTPLOAD_USERADT`. The function returns null if `dest_pload_type` parameter is none of these three constants.

This function assumes that the J2EE compliant mode is true.

Syntax

```
Function convert_jms_selector(
   selector IN VARCHAR2,
   dest_pload_type IN PLS_INTEGER)
RETURN VARCHAR2
```

Returns

The converted Oracle Database Advanced Queuing rule or null if there is any conversion error.

Exceptions

ORA-24197 if the Java stored procedure throws an exception during execution.

Convert with Destination Payload Type and Compliant Mode Specified

The third procedure takes a `dest_pload_type` parameter and a `compliant` parameter. The conversion of a JMS selector to an Oracle Database Advanced Queuing rule happens only if the `dest_pload_type` parameter is `SYS.dbms_jms_plsql.DESTPLOAD_JMSTYPE` or `SYS.dbms_jms_plsql.DESTPLOAD_ANYDATA`. The function returns exactly the same `VARCHAR2` value as the selector parameter if the `dest_pload_type` parameter is `SYS.dbms_jms_plsql.DESTPLOAD_USERADT`. The function returns null if the `dest_pload_type` parameter is none of these three constants.

The `compliant` parameter controls if the conversion is in J2EE compliant mode or not. The noncompliant conversion of a JMS selector is for backward compatibility.

Syntax

```
Function convert_jms_selector(
   selector         IN  VARCHAR2,
   dest_pload_type  IN  PLS_INTEGER,
   compliant        IN  BOOLEAN )
```

Returns

The converted Oracle Database Advanced Queuing rule or null if there is any conversion error.

Exceptions

ORA-24197 if the Java stored procedure throws an exception during execution.

Summary of JMS Types

This lists shows the JMS types.

- SYS.AQ$_JMS_MESSAGE Type
- SYS.AQ$_JMS_TEXT_MESSAGE Type
- SYS.AQ$_JMS_BYTES_MESSAGE Type
- SYS.AQ$_JMS_MAP_MESSAGE Type
- SYS.AQ$_JMS_STREAM_MESSAGE Type
- SYS.AQ$_JMS_OBJECT_MESSAGE Type
- SYS.AQ$_JMS_NAMEARRAY Type
- SYS.AQ$_JMS_VALUE Type
- SYS.AQ$_JMS_EXCEPTION Type

SYS.AQ$_JMS_MESSAGE Type

This ADT type can represent any of five different JMS message types: text message, bytes message, stream message, map message, or object message. Queues created using this ADT can therefore store all five types of JMS messages.

This section contains these topics:

- CONSTRUCT Static Functions
- Cast Methods
- JMS Header Methods
- System Properties Methods
- User Properties Methods
- Payload Methods

Syntax

```
TYPE AQ$_JMS_MESSAGE AS OBJECT(
 header         aq$_jms_header,
 senderid       varchar2(100),
 message_type   INT,
 text_len       INT,
bytes_len     INT,
text_vc       varchar2(4000),
bytes_raw     raw(2000),
text_lob      clob,
bytes_lob     blob,
STATIC FUNCTION  construct (mtype      IN  INT)
  RETURN aq$_jms_message,
STATIC FUNCTION  construct (text_msg   IN  aq$_jms_text_message)
  RETURN aq$_jms_message,
STATIC FUNCTION  construct (bytes_msg  IN  aq$_jms_bytes_message)
  RETURN aq$_jms_message,
STATIC FUNCTION  construct (stream_msg IN  aq$_jms_stream_message)
  RETURN aq$_jms_message,
STATIC FUNCTION  construct (map_msg    IN  aq$_jms_map_message)
  RETURN aq$_jms_message,
STATIC FUNCTION  construct (object_msg IN  aq$_jms_object_message)
  RETURN aq$_jms_message,
MEMBER FUNCTION  cast_to_bytes_msg  RETURN aq$_jms_bytes_message,
MEMBER FUNCTION  cast_to_map_msg    RETURN aq$_jms_map_message,
MEMBER FUNCTION  cast_to_object_msg RETURN aq$_jms_object_message,
MEMBER FUNCTION  cast_to_stream_msg RETURN aq$_jms_stream_message,
MEMBER FUNCTION  cast_to_text_msg   RETURN aq$_jms_text_message,
MEMBER PROCEDURE set_replyto  (replyto IN sys.aq$_agent),
MEMBER PROCEDURE set_type     (type     IN  VARCHAR),
MEMBER PROCEDURE set_userid   (userid   IN  VARCHAR),
MEMBER PROCEDURE set_appid    (appid    IN  VARCHAR),
MEMBER PROCEDURE set_groupid  (groupid  IN  VARCHAR),
MEMBER PROCEDURE set_groupseq (groupseq IN  INT),
MEMBER FUNCTION  get_replyto  RETURN sys.aq$_agent,
MEMBER FUNCTION  get_type     RETURN VARCHAR,
MEMBER FUNCTION  get_userid   RETURN VARCHAR,
MEMBER FUNCTION  get_appid    RETURN VARCHAR,
MEMBER FUNCTION  get_groupid  RETURN VARCHAR,
MEMBER FUNCTION  get_groupseq RETURN INT,
MEMBER PROCEDURE clear_properties,
MEMBER PROCEDURE set_boolean_property (property_name IN VARCHAR,
  property_value IN BOOLEAN),
MEMBER PROCEDURE set_byte_property    (property_name IN VARCHAR,
  property_value IN INT),
MEMBER PROCEDURE set_double_property  (property_name IN VARCHAR,
  property_value IN DOUBLE PRECISION),
MEMBER PROCEDURE set_float_property   (property_name IN VARCHAR,
  property_value IN FLOAT),
MEMBER PROCEDURE set_int_property     (property_name IN VARCHAR,
  property_value IN INT),
MEMBER PROCEDURE set_long_property    (property_name IN VARCHAR,
  property_value IN NUMBER),
MEMBER PROCEDURE set_short_property   (property_name IN VARCHAR,
  property_value IN INT),
MEMBER PROCEDURE set_string_property  (property_name IN VARCHAR,
  property_value IN VARCHAR),
MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
MEMBER FUNCTION get_byte_property    (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_double_property  (property_name IN VARCHAR)
  RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
MEMBER FUNCTION get_int_property     (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
MEMBER FUNCTION get_short_property   (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_string_property  (property_name IN VARCHAR) RETURN VARCHAR,
MEMBER PROCEDURE set_text     (payload  IN  VARCHAR2),
MEMBER PROCEDURE set_text     (payload  IN  CLOB),
 MEMBER PROCEDURE set_bytes     (payload  IN  RAW),
 MEMBER PROCEDURE set_bytes     (payload  IN  BLOB),
 MEMBER PROCEDURE get_text      (payload  OUT VARCHAR2),
 MEMBER PROCEDURE get_text      (payload  OUT CLOB),
 MEMBER PROCEDURE get_bytes     (payload  OUT RAW),
 MEMBER PROCEDURE get_bytes     (payload  OUT BLOB));
```

CONSTRUCT Static Functions

There are six `CONSTRUCT` static functions in this type.

STATIC FUNCTION construct (mtype IN INT) RETURN aq$_jms_message
Creates an instance of `aq$_jms_message`, which can hold a specific type of JMS message (TextMessage, BytesMessage, MapMessage, StreamMessage or ObjectMessage). The message type of the created `aq$_jms_message` instance depends on the `mtype` parameter passed to the construct method. Once a message has been constructed, it can be used to store JMS messages of the type it has been constructed to hold.
The `mtype` parameter must be one of the following constants described in "Oracle JMS Type Constants":

```
DBMS_AQ.JMS_TEXT_MESSAGE
DBMS_AQ.JMS_BYTES_MESSAGE
DBMS_AQ.JMS_STREAM_MESSAGE
DBMS_AQ.JMS_MAP_MESSAGE
DBMS_AQ.JMS_OBJECT_MESSAGE
```

STATIC FUNCTION construct (text_msg IN aq$_jms_text_message) RETURN aq$_jms_message
Creates an `aq$_jms_message` from an `aq$_jms_text_message`.

STATIC FUNCTION construct (bytes_msg IN aq$_jms_bytes_message) RETURN aq$_jms_message;
Creates an `aq$_jms_message` from an `aq$_jms_bytes_message`.

STATIC FUNCTION construct (stream_msg IN aq$_jms_stream_message) RETURN aq$_jms_message;
Creates an `aq$_jms_message` from an `aq$_jms_stream_message`.

STATIC FUNCTION construct (map_msg IN aq$_jms_map_message) RETURN aq$_jms_message;
Creates an `aq$_jms_message` from an `aq$_jms_map_message`.

STATIC FUNCTION construct (object_msg IN aq$_jms_object_message) RETURN aq$_jms_message;
Creates an `aq$_jms_message` from an `aq$_jms_object_message`.

Cast Methods

cast_to_bytes_msg RETURN aq$_jms_bytes_message
Casts an `aq$_jms_message` to an `aq$_jms_bytes_message`. Returns an `aq$_jms_bytes_message` or null if the `message_type` attribute of the `aq$_jms_message` is not `DBMS_AQ.JMS_BYTES_MESSAGE`. This function raises ORA-24198 if the `message_type` field of the `aq$_jms_message` is not `DBMS_AQJMS.JMS_BYTES_MESSAGE`.

cast_to_map_msg RETURN aq$_jms_map_message
Casts an `aq$_jms_message` to an `aq$_jms_map_message`. Returns an `aq$_jms_map_message` or null if the `message_type` attribute of the `aq$_jms_message` is not `DBMS_AQ.JMS_MAP_MESSAGE`.

This function raises ORA-24198 if the `message_type` field of the `aq$_jms_message` is not `DBMS_AQJMS.JMS_MAP_MESSAGE`.

cast_to_object_msg RETURN aq$_jms_object_message
Casts an `aq$_jms_message` to an `aq$_jms_object_message`. Returns an `aq$_jms_object_message` or null if the `message_type` attribute of the `aq$_jms_message` is not `DBMS_AQ.JMS_OBJECT_MESSAGE`. This function raises ORA-24198 if the `message_type` field of the `aq$_jms_message` is not `DBMS_AQJMS.JMS_OBJECT_MESSAGE`.

cast_to_stream_msg RETURN aq$_jms_stream_message
Casts an `aq$_jms_message` to an `aq$_jms_stream_message`. Returns an `aq$_jms_stream_message` or null if the `message_type` attribute of the `aq$_jms_message` is not `DBMS_AQ.JMS_STREAM_MESSAGE`. This function raises ORA-24198 if the `message_type` field of the `aq$_jms_message` is not `DBMS_AQJMS.JMS_STREAM_MESSAGE`.

cast_to_text_msg RETURN aq$_jms_text_message
Casts an `aq$_jms_message` to an `aq$_jms_text_message`. Returns an `aq$_jms_text_message` or null if the `message_type` attribute of the `aq$_jms_message` is not `DBMS_AQ.JMS_TEXT_MESSAGE`. This function raises ORA-24198 if the `message_type` field of the `aq$_jms_message` is not `DBMS_AQJMS.JMS_TEXT_MESSAGE`.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the `replyto` parameter, which corresponds to `JMSReplyTo`.

get_replyto RETURN sys.aq$_agent
Returns `replyto`, which corresponds to `JMSReplyTo`.

set_type (type IN VARCHAR)
Sets the JMS type, which can be any text and corresponds to `JMSType`.

get_type RETURN VARCHAR
Returns `type`, which corresponds to `JMSType`.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets `userid`, which corresponds to `JMSXUserID`.

set_appid (appid IN VARCHAR)
Sets `appid`, which corresponds to `JMSXAppID`.

set_groupid (groupid IN VARCHAR)
Sets `groupid`, which corresponds to `JMSXGroupID`.

set_groupseq (groupseq IN INT)
Sets `groupseq`, which corresponds to `JMSXGroupSeq`.

get_userid RETURN VARCHAR
Returns `userid`, which corresponds to `JMSXUserID`.

get_appid RETURN VARCHAR
Returns `appid`, which corresponds to `JMSXAppID`.

get_groupid RETURN VARCHAR
Returns `groupid`, which corresponds to `JMSXGroupID`.

get_groupseq RETURN VARCHAR
Returns `groupseq`, which corresponds to `JMSXGroupSeq`.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value` in an internal representation (a `NUMBER` type). Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether property_name is null or exists. If it is not null, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `byte` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If it is not null, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because the `INT` datatype is 38 bits in PL/SQL and Oracle Database. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the long datatype is 64 bits. Therefore, no range check is needed. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If it is not null, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether `property_name` is null or exists. If it is not null, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a `long` property, then this function returns the value of the property. Otherwise it returns a null.

get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

get_string_property (property_name IN VARCHAR) RETURN VARCHAR
If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

Payload Methods

set_text (payload IN VARCHAR2)
Sets the payload, a `VARCHAR2` value, to an internal representation.

set_text (payload IN CLOB),
Sets the payload, a `CLOB` value, to an internal representation.

set_bytes (payload IN RAW)
Sets the payload, a `RAW` value, to an internal representation.

set_bytes (payload IN BLOB)
Sets the payload, a `BLOB` value, to an internal representation.

get_text (payload OUT VARCHAR2)
Puts the internal representation of the payload into a `VARCHAR2` variable payload.

get_text (payload OUT CLOB)
Puts the internal representation of the payload into a `CLOB` variable payload.

get_bytes (payload OUT RAW)
Puts the internal representation of the payload into a `RAW` variable payload.

get_bytes (payload OUT BLOB)
Puts the internal representation of the payload into a `BLOB` variable payload.

SYS.AQ$_JMS_TEXT_MESSAGE Type

This type is the ADT used to store a `TextMessage` in an Oracle Database Advanced Queuing queue.

This section contains these topics:

- CONSTRUCT aq$_jms_text_message Function
- JMS Header Methods
- System Properties Methods
- User Properties Methods
- Payload Methods

Syntax

```
TYPE AQ$_JMS_TEXT_MESSAGE AS OBJECT(
 header    aq$_jms_header,
 text_len  INT,
 text_vc   varchar2(4000),
 text_lob  clob,
 STATIC FUNCTION construct    RETURN aq$_jms_text_message,
 MEMBER PROCEDURE set_replyto  (replyto  IN  sys.aq$_agent),
 MEMBER PROCEDURE set_type     (type     IN  VARCHAR),
 MEMBER FUNCTION  get_replyto RETURN sys.aq$_agent,
 MEMBER FUNCTION  get_type    RETURN VARCHAR,
 MEMBER PROCEDURE set_userid   (userid   IN  VARCHAR),
 MEMBER PROCEDURE set_appid    (appid    IN  VARCHAR),
 MEMBER PROCEDURE set_groupid  (groupid  IN  VARCHAR),
 MEMBER PROCEDURE set_groupseq (groupseq IN  INT),
 MEMBER FUNCTION get_userid   RETURN VARCHAR,
 MEMBER FUNCTION get_appid    RETURN VARCHAR,
 MEMBER FUNCTION get_groupid  RETURN VARCHAR,
 MEMBER FUNCTION get_groupseq RETURN INT,
 MEMBER PROCEDURE clear_properties,
 MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR,
   property_value IN BOOLEAN),
 MEMBER PROCEDURE set_byte_property   (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_double_property (property_name IN VARCHAR,
   property_value IN DOUBLE PRECISION),
 MEMBER PROCEDURE set_float_property  (property_name IN VARCHAR,
   property_value IN FLOAT),
 MEMBER PROCEDURE set_int_property    (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_long_property   (property_name IN VARCHAR,
   property_value IN NUMBER),
 MEMBER PROCEDURE set_short_property  (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
   property_value IN VARCHAR),
 MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR)
   RETURN BOOLEAN,
MEMBER FUNCTION get_byte_property    (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_double_property  (property_name IN VARCHAR)
  RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
MEMBER FUNCTION get_int_property     (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
MEMBER FUNCTION get_short_property   (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_string_property  (property_name IN VARCHAR)
  RETURN VARCHAR,
MEMBER PROCEDURE set_text            (payload  IN  VARCHAR2),
MEMBER PROCEDURE set_text            (payload  IN  CLOB),
MEMBER PROCEDURE get_text            (payload  OUT VARCHAR2),
MEMBER PROCEDURE get_text            (payload  OUT CLOB));
```

CONSTRUCT aq$_jms_text_message Function

STATIC FUNCTION construct RETURN aq$_jms_text_message
Creates an empty `aq$_jms_text_message`.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the `replyto` parameter, which corresponds to `JMSReplyTo` in JMS.

set_type (type IN VARCHAR)
Sets the JMS type, which can be any text, and which corresponds to `JMSType` in JMS.

get_replyto RETURN sys.aq$_agent
Returns `replyto`, which corresponds to `JMSReplyTo`.

get_type RETURN VARCHAR
Returns `type`, which corresponds to `JMSType`.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets `userid`, which corresponds to `JMSXUserID` in JMS.

set_appid (appid IN VARCHAR)
Sets `appid`, which corresponds to `JMSXAppID` in JMS.

set_groupid (groupid IN VARCHAR)
Sets `groupid`, which corresponds to `JMSXGroupID` in JMS.

set_groupseq (groupseq IN INT)
Sets `groupseq`, which corresponds to `JMSXGroupSeq` in JMS.

get_userid RETURN VARCHAR
Returns `userid`, which corresponds to `JMSXUserID`.

get_appid RETURN VARCHAR
Returns `appid`, which corresponds to `JMSXAppID`.

get_groupid RETURN VARCHAR
Returns `groupid`, which corresponds to `JMSXGroupID`.

get_groupseq RETURN INT
Returns `groupseq`, which corresponds to `JMSXGroupSeq`.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value` in an internal representation. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `BYTE` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the `INT` datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the `long` datatype is 64 bits. Therefore, no range check is needed.Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a `long` property, then this function returns the value of the property. Otherwise it returns a null.

get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

get_string_property (property_name IN VARCHAR) RETURN VARCHAR)
If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

Payload Methods

set_text (payload IN VARCHAR2)
Sets the payload, a `VARCHAR2` value, to an internal representation.

set_text (payload IN CLOB)
Sets the payload, a `CLOB` value, to an internal representation.

get_text (payload OUT VARCHAR2)
Puts the internal representation of the payload into a `VARCHAR2` variable payload.

get_text (payload OUT CLOB)
Puts the internal representation of the payload into a `CLOB` variable payload.

SYS.AQ$_JMS_BYTES_MESSAGE Type

The `SYS.AQ$_JMS_BYTES_MESSAGE` type is the ADT used to store a `BytesMessage` in an Oracle Database Advanced Queuing queue.

This section contains these topics:

- CONSTRUCT aq$_jms_bytes_message Function
- JMS Header Methods
- System Properties Methods

- User Properties Methods
- Payload Methods

Syntax

```
TYPE AQ$_JMS_BYTES_MESSAGE AS OBJECT(
 header      aq$_jms_header,
 bytes_len   INT,
 bytes_raw   raw(2000),
 bytes_lob   blob,
 STATIC FUNCTION construct RETURN aq$_jms_bytes_message,
 MEMBER PROCEDURE set_replyto  (replyto IN sys.aq$_agent),
 MEMBER PROCEDURE set_type     (type    IN VARCHAR),
 MEMBER FUNCTION get_replyto RETURN sys.aq$_agent,
 MEMBER FUNCTION get_type    RETURN VARCHAR,
 MEMBER PROCEDURE set_userid   (userid   IN VARCHAR),
 MEMBER PROCEDURE set_appid    (appid    IN VARCHAR),
 MEMBER PROCEDURE set_groupid  (groupid  IN VARCHAR),
 MEMBER PROCEDURE set_groupseq (groupseq IN INT),
 MEMBER FUNCTION get_userid   RETURN VARCHAR,
 MEMBER FUNCTION get_appid    RETURN VARCHAR,
 MEMBER FUNCTION get_groupid  RETURN VARCHAR,
 MEMBER FUNCTION get_groupseq RETURN INT,
 MEMBER PROCEDURE clear_properties,
 MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR,
   property_value IN BOOLEAN),
 MEMBER PROCEDURE set_byte_property   (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_double_property (property_name IN VARCHAR,
   property_value IN DOUBLE PRECISION),
 MEMBER PROCEDURE set_float_property  (property_name IN VARCHAR,
   property_value IN FLOAT),
 MEMBER PROCEDURE set_int_property    (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_long_property   (property_name IN VARCHAR,
   property_value IN NUMBER),
 MEMBER PROCEDURE set_short_property  (property_name IN VARCHAR,
   property_valuE IN INT),
 MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
   property_value IN VARCHAR),
 MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
 MEMBER FUNCTION get_byte_property    (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_double_property  (property_name IN VARCHAR)
   RETURN DOUBLE PRECISION,
 MEMBER FUNCTION get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
 MEMBER FUNCTION get_int_property     (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
 MEMBER FUNCTION get_short_property   (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_string_property  (property_name IN VARCHAR) RETURN VARCHAR,
 MEMBER PROCEDURE set_bytes           (payload  IN RAW),
 MEMBER PROCEDURE set_bytes           (payload  IN BLOB),
 MEMBER PROCEDURE get_bytes           (payload  OUT RAW),
 MEMBER PROCEDURE get_bytes           (payload  OUT BLOB),
 MEMBER FUNCTION  prepare             (id IN PLS_INTEGER) RETURN PLS_INTEGER,
 MEMBER PROCEDURE reset               (id IN PLS_INTEGER),
 MEMBER PROCEDURE flush               (id IN PLS_INTEGER),
 MEMBER PROCEDURE clear_body          (id IN PLS_INTEGER),
 MEMBER PROCEDURE clean               (id IN PLS_INTEGER),
 STATIC PROCEDURE clean_all,
 MEMBER FUNCTION get_mode            (id IN PLS_INTEGER) RETURN PLS_INTEGER,
 MEMBER FUNCTION read_boolean        (id IN PLS_INTEGER) RETURN BOOLEAN,
 MEMBER FUNCTION read_byte           (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_bytes         (id IN PLS_INTEGER,
  value OUT NOCOPY BLOB, length IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_char          (id IN PLS_INTEGER) RETURN CHAR,
MEMBER FUNCTION read_double        (id IN PLS_INTEGER) RETURN DOUBLE PRECISION,
MEMBER FUNCTION read_float         (id IN PLS_INTEGER) RETURN FLOAT,
MEMBER FUNCTION read_int           (id IN PLS_INTEGER) RETURN INT,
MEMBER FUNCTION read_long          (id IN PLS_INTEGER) RETURN NUMBER,
MEMBER FUNCTION read_short         (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_unsigned_byte  (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION read_unsigned_short (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE read_utf          (id IN PLS_INTEGER, value OUT NOCOPY CLOB),
MEMBER PROCEDURE write_boolean     (id IN PLS_INTEGER, value IN BOOLEAN),
MEMBER PROCEDURE write_byte        (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_bytes       (id IN PLS_INTEGER, value IN RAW),
MEMBER PROCEDURE write_bytes       (id IN PLS_INTEGER, value IN BLOB),
MEMBER PROCEDURE write_bytes       (id IN PLS_INTEGER, value IN RAW,
  offset IN PLS_INTEGER, length IN PLS_INTEGER),
MEMBER PROCEDURE write_bytes       (id IN PLS_INTEGER, value IN BLOB,
  offset IN INT, length IN INT),
MEMBER PROCEDURE write_char        (id IN PLS_INTEGER, value IN CHAR),
MEMBER PROCEDURE write_double      (id IN PLS_INTEGER,
  value IN DOUBLE PRECISION),
MEMBER PROCEDURE write_float       (id IN PLS_INTEGER, value IN FLOAT),
MEMBER PROCEDURE write_int         (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_long        (id IN PLS_INTEGER, value IN NUMBER),
MEMBER PROCEDURE write_short       (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_utf         (id IN PLS_INTEGER, value IN VARCHAR2),
MEMBER PROCEDURE write_utf         (id IN PLS_INTEGER, value IN CLOB));
```

CONSTRUCT aq$_jms_bytes_message Function

STATIC FUNCTION construct RETURN aq$_jms_bytes_message
Creates an empty `aq$_jms_bytes_message`.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the `replyto` parameter, which corresponds to `JMSReplyTo` in JMS.

set_type (type IN VARCHAR)
Sets the JMS type, which can be any text, and which corresponds to `JMSType` in JMS.

get_replyto RETURN sys.aq$_agent
Returns `replyto`, which corresponds to `JMSReplyTo`.

get_type RETURN VARCHAR
Returns `type`, which corresponds to `JMSType`.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets `userid`, which corresponds to `JMSXUserID` in JMS.

set_appid (appid IN VARCHAR)
Sets `appid`, which corresponds to `JMSXAppID` in JMS.

set_groupid (groupid IN VARCHAR)
Sets `groupid`, which corresponds to `JMSXGroupID` in JMS.

set_groupseq (groupseq IN INT)
Sets `groupseq`, which corresponds to `JMSXGroupSeq` in JMS.

get_userid RETURN VARCHAR
Returns `userid`, which corresponds to `JMSXUserID`.

get_appid RETURN VARCHAR
Returns `appid`, which corresponds to `JMSXAppID`.

get_groupid RETURN VARCHAR
Returns `groupid`, which corresponds to `JMSXGroupID`.

get_groupseq RETURN NUMBER
Returns `groupseq`, which corresponds to `JMSXGroupSeq`.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value` in an internal representation. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `BYTE` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the `INT` datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the `long` datatype is 64 bits. Therefore, no range check is needed.Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a `long` property, then this function returns the value of the property. Otherwise it returns a null.

get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

get_string_property (property_name IN VARCHAR) RETURN VARCHAR
If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

Payload Methods

set_bytes (payload in RAW)
Sets the payload, a `RAW` value, to an internal representation.

set_bytes (payload in BLOB)
Sets the payload, a `BLOB` value, to an internal representation.

get_bytes (payload out RAW)
Puts the internal representation of the payload into a `RAW` variable payload. Raises exception ORA-24190 if the length of the internal payload is more than 32767 (the maximum length of `RAW` in PL/SQL).

get_bytes (payload out BLOB)
Puts the internal representation of the payload into a `BLOB` variable payload.

prepare (id IN PLS_INTEGER) RETURN PLS_INTEGER
Takes the byte array stored in aq$_jms_bytes_message and decodes it as a Java object in the Java stored procedure. The result of the decoding is stored as a static variable in Jserv session memory. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, then a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.
This function also sets the message access mode to `MESSAGE_ACCESS_READONLY`. Subsequent calls of `write_XXX` procedure raise an ORA-24196 error. Users can call the `clear_body` procedure to set the message access mode to `MESSAGE_ACCESS_READONLY`.
This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

reset (id IN PLS_INTEGER)
Resets the starting position of the stream to the beginning and puts the bytes message in read-only mode. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

flush (id IN PLS_INTEGER)
Takes the static variable in Jserv and synchronizes the content back to the `aq$_jms_bytes_message`. This procedure will not affect the underlying access mode. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clear_body (id IN PLS_INTEGER)
Sets the Java stored procedure static variable to empty payload. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.
It also sets the message access mode to `MESSAGE_ACCESS_WRITEONLY`. Later calls of `read_XXX` procedure raise ORA-24196 error. Users can call the `reset` or `prepare` procedures to set the message access mode to `MESSAGE_ACCESS_READONLY`. Write-only and read-only modes affect only the payload functions of `AQ$_JMS_BYTES_MESSAGE`. They do not affect the header functions.
This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

clean (id IN PLS_INTEGER)
Closes and cleans up the `DataInputStream` or `DataOutputStream` at the Java stored procedure side corresponding to the operation ID. It is very important to call this procedure to avoid memory leaks. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clean_all
Closes and cleans up all the messages in the corresponding type of message store at the Java stored procedure side. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution.

get_mode (id IN PLS_INTEGER) RETURN PLS_INTEGER
Returns the current mode of this message. The return value is either `SYS.dbms_jms.plsql.MESSAGE_ACCESS_READONLY` or

`SYS.dbms_jms.plsql.MESSAGE_ACCESS_WRITEONLY`. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

read_boolean (id IN PLS_INTEGER) RETURN BOOLEAN
Reads a Boolean value from the bytes message and returns the Boolean value read. Null is returned if the end of the message stream has been reached. Parameter `id` is the operation ID. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_byte (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads a `BYTE` value from the bytes message and returns the `BYTE` value read. Null is returned if the end of the stream has been reached. Because there is no `BYTE` type in PL/SQL, Oracle Database uses `PLS_INTEGER` to represent a `BYTE`. Although PL/SQL users get a `PLS_INTEGER`, they are guaranteed that the value is in the Java `BYTE` value range. If this value is issued with a `write_byte` function, then there will not be an out of range error. Parameter `id` is the operation ID. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_bytes (id IN PLS_INTEGER, value OUT NO COPY BLOB, length IN PLS_INTEGER) RETURN PLS_INTEGER
Reads length of the bytes from bytes message stream into value and returns the total number of bytes read. If there is no more data (because the end of the stream has been reached), then it returns -1. Raises exceptions ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_char (id IN PLS_INTEGER) RETURN CHAR
Reads a character value from the bytes message and returns the character value read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_double (id IN PLS_INTEGER) RETURN DOUBLE PRECISION
Reads a double from the bytes message and returns the character value read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_float (id IN PLS_INTEGER) RETURN FLOAT
Reads a float from the bytes message and returns the float read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_int (id IN PLS_INTEGER) RETURN INT
Reads an `INT` from the bytes message and returns the `INT` read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_long (id IN PLS_INTEGER) RETURN NUMBER
Reads a long from the bytes message and returns the long read. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-

only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_short (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads a short value from the bytes message and returns the short value read. Null is returned if the end of the stream has been reached. Because there is no short type in PL/SQL, `PLS_INTEGER` is used to represent a `SHORT`. Although PL/SQL users get an `PLS_INTEGER`, they are guaranteed that the value is in the Java short value range. If this value is issued with a `write_short` function, then there will not be an out of range error. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_unsigned_byte (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads an unsigned 8-bit number from the bytes message stream and returns the next byte from the bytes message stream, interpreted as an unsigned 8-bit number. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_unsigned_short (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads an unsigned 16-bit number from the bytes message stream and returns the next two bytes from the bytes message stream, interpreted as an unsigned 16-bit integer. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_utf (id IN PLS_INTEGER, value OUT NOCOPY CLOB)
Reads a string that has been encoded using a UTF-8 format from the bytes message. Null is returned if the end of the stream has been reached. Raises exception ORA-24196 if the bytes message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_boolean (id IN PLS_INTEGER, value IN BOOLEAN)
Writes a Boolean to the bytes message stream as a 1-byte value. The value `true` is written as the value (byte)1. The value `false` is written as the value (byte)0. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_byte (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes a byte to the bytes message. Because there is no `BYTE` type in PL/SQL, `PLS_INTEGER` is used to represent a `BYTE`. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN RAW)
Writes an array of bytes to the bytes message. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN BLOB)
Writes an array of bytes to the bytes message. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN RAW, offset IN PLS_INTEGER, length IN PLS_INTEGER)
Writes a portion of a byte array to the bytes message stream. Parameter `offset` is the initial offset within the byte array. If the range [offset, offset+length] exceeds the boundary of the byte array value, then a Java IndexOutOfBounds exception is thrown in the Java stored procedure and this procedure raises error ORA-24197. The index starts from 0. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN BLOB, offset IN INT, length IN INT)
Writes a portion of a byte array to the bytes message stream. Parameter `offset` is the initial offset within the byte array. If the range [offset, offset+length] exceeds the boundary of the byte array value, then a Java IndexOutOfBounds exception is thrown in the Java stored procedure and this procedure raises error ORA-24197. The index starts from 0. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_char (id IN PLS_INTEGER, value IN CHAR)
Writes a character value to the bytes message. If this value has multiple characters, it is the first character that is written. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_double (id IN PLS_INTEGER, value IN DOUBLE PRECISION)
Writes a double to the bytes message. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_float (id IN PLS_INTEGER, value IN FLOAT)
Writes a float to the bytes message. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_int (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes an `INT` to the bytes message. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_long (id IN PLS_INTEGER, value IN NUMBER)
Writes a long to the bytes message. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_short (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes a short to the bytes message as two bytes, high byte first. Because there is no short type in PL/SQL, `INT` is used to represent a short. Raises exception ORA-24193 if the parameter value exceeds the valid range, ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_utf (id IN PLS_INTEGER, value IN VARCHAR2)
Writes a string to the bytes message stream using UTF-8 encoding in a machine-independent manner. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197

if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_utf (id IN PLS_INTEGER, value IN CLOB)
Writes a string to the bytes message stream using UTF-8 encoding in a machine-independent manner. Raises exception ORA-24196 if the bytes message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

SYS.AQ$_JMS_MAP_MESSAGE Type

This type is the ADT used to store a `MapMessage` in an Oracle Database Advanced Queuing queue.

This section contains these topics:

- CONSTRUCT aq$_jms_map_message Function
- JMS Header Methods
- System Properties Methods
- User Properties Methods
- Payload Methods

Syntax

```
TYPE aq$_jms_map_message AS object(
 header     aq$_jms_header,
 bytes_len  int,
 bytes_raw  raw(2000),
 bytes_lob  blob,
 STATIC FUNCTION  construct   RETURN aq$_jms_map_message,
 MEMBER PROCEDURE set_replyto  (replyto IN sys.aq$_agent),
 MEMBER PROCEDURE set_type     (type    IN VARCHAR),
 MEMBER FUNCTION get_replyto  RETURN sys.aq$_agent,
 MEMBER FUNCTION get_type     RETURN VARCHAR,
 MEMBER PROCEDURE set_userid   (userid   IN VARCHAR),
 MEMBER PROCEDURE set_appid    (appid    IN VARCHAR),
 MEMBER PROCEDURE set_groupid  (groupid  IN VARCHAR),
 MEMBER PROCEDURE set_groupseq (groupseq IN INT),
 MEMBER FUNCTION get_userid   RETURN VARCHAR,
 MEMBER FUNCTION get_appid    RETURN VARCHAR,
 MEMBER FUNCTION get_groupid  RETURN VARCHAR,
 MEMBER FUNCTION get_groupseq RETURN INT,
 MEMBER PROCEDURE clear_properties,
 MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR,
   property_value IN BOOLEAN),
 MEMBER PROCEDURE set_byte_property   (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_double_property (property_name IN VARCHAR,
   property_value IN DOUBLE PRECISION),
 MEMBER PROCEDURE set_float_property  (property_name IN VARCHAR,
   property_value IN FLOAT),
 MEMBER PROCEDURE set_int_property    (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_long_property   (property_name IN VARCHAR,
   property_value IN NUMBER),
 MEMBER PROCEDURE set_short_property  (property_name IN VARCHAR,
   property_valuE IN INT),
 MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
  property_value IN VARCHAR),
MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
MEMBER FUNCTION get_byte_property    (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_double_property  (property_name IN VARCHAR)
  RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
MEMBER FUNCTION get_int_property     (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
MEMBER FUNCTION get_short_property   (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_string_property  (property_name IN VARCHAR) RETURN VARCHAR,
MEMBER PROCEDURE set_bytes   (payload IN RAW),
MEMBER PROCEDURE set_bytes   (payload IN BLOB),
MEMBER PROCEDURE get_bytes   (payload OUT RAW),
MEMBER PROCEDURE get_bytes   (payload OUT BLOB),
MEMBER FUNCTION  prepare     (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE flush       (id  IN  PLS_INTEGER),
MEMBER PROCEDURE clear_body  (id IN PLS_INTEGER),
MEMBER PROCEDURE clean       (id  IN  PLS_INTEGER),
STATIC PROCEDURE clean_all,
MEMBER PROCEDURE set_boolean (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN BOOLEAN),
MEMBER PROCEDURE set_byte    (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  PLS_INTEGER),
MEMBER PROCEDURE set_bytes   (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  RAW),
MEMBER PROCEDURE set_bytes   (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN RAW, offset IN INT, length IN INT),
MEMBER PROCEDURE set_bytes   (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN BLOB),
MEMBER PROCEDURE set_bytes   (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN BLOB, offset IN INT, length IN INT),
MEMBER PROCEDURE set_char    (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  CHAR),
MEMBER PROCEDURE set_double  (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  DOUBLE PRECISION),
MEMBER PROCEDURE set_float   (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  FLOAT),
MEMBER PROCEDURE set_int     (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  PLS_INTEGER),
MEMBER PROCEDURE set_long    (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  NUMBER),
MEMBER PROCEDURE set_short   (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  PLS_INTEGER),
MEMBER PROCEDURE set_string  (id IN PLS_INTEGER, name IN VARCHAR2,
  value IN  VARCHAR2),
MEMBER PROCEDURE set_string  (id IN  PLS_INTEGER, name IN VARCHAR2,
  value IN  CLOB),
MEMBER FUNCTION get_boolean  (id IN PLS_INTEGER, name IN VARCHAR2)
  RETURN BOOLEAN,
MEMBER FUNCTION  get_byte    (id IN PLS_INTEGER, name IN VARCHAR2)
  RETURN PLS_INTEGER,
MEMBER PROCEDURE get_bytes   (id IN PLS_INTEGER, name IN VARCHAR2,
  value OUT NOCOPY BLOB),
MEMBER FUNCTION get_char     (id IN PLS_INTEGER, name IN VARCHAR2) RETURN CHAR,
MEMBER FUNCTION get_double   (id IN PLS_INTEGER, name IN VARCHAR2)
  RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float   (id IN PLS_INTEGER, name IN VARCHAR2) RETURN FLOAT,
MEMBER FUNCTION get_int     (id IN PLS_INTEGER, name IN VARCHAR2)
  RETURN PLS_INTEGER,
MEMBER FUNCTION get_long    (id IN PLS_INTEGER, name IN VARCHAR2)
  RETURN NUMBER,
MEMBER FUNCTION get_short   (id IN PLS_INTEGER, name IN VARCHAR2)
```

```
  RETURN PLS_INTEGER,
MEMBER PROCEDURE get_string (id IN PLS_INTEGER, name IN VARCHAR2,
  value OUT NOCOPY CLOB),
MEMBER FUNCTION get_names   (id IN PLS_INTEGER) RETURN aq$_jms_namearray,
MEMBER FUNCTION get_names   (id IN PLS_INTEGER, names OUT aq$_jms_namearray,
  offset IN  PLS_INTEGER, length IN  PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE get_object (id IN PLS_INTEGER, name IN  VARCHAR2,
  value  OUT NOCOPY AQ$_JMS_VALUE),
MEMBER FUNCTION get_size    (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION item_exists (id IN PLS_INTEGER, name IN VARCHAR2)
  RETURN BOOLEAN);
```

CONSTRUCT aq$_jms_map_message Function

STATIC FUNCTION construct RETURN aq$_jms_map_message
Creates an empty `aq$_jms_map_message` object.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the `replyto` parameter, which corresponds to `JMSReplyTo` in JMS.

set_type (type IN VARCHAR)
Sets the JMS type, which can be any text, and which corresponds to `JMSType` in JMS.

get_replyto RETURN sys.aq$_agent
Returns `replyto`, which corresponds to `JMSReplyTo`.

get_type RETURN VARCHAR
Returns `type`, which corresponds to `JMSType`.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets `userid`, which corresponds to `JMSXUserID` in JMS.

set_appid (appid IN VARCHAR)
Sets `appid`, which corresponds to `JMSXAppID` in JMS.

set_groupid (groupid IN VARCHAR)
Sets `groupid`, which corresponds to `JMSXGroupID` in JMS.

set_groupseq (groupseq IN INT)
Sets `groupseq`, which corresponds to `JMSXGroupSeq` in JMS.

get_userid RETURN VARCHAR
Returns `userid`, which corresponds to `JMSXUserID`.

get_appid RETURN VARCHAR
Returns `appid`, which corresponds to `JMSXAppID`.

get_groupid RETURN VARCHAR
Returns `groupid`, which corresponds to `JMSXGroupID`.

get_groupseq RETURN NUMBER
Returns `groupseq`, which corresponds to `JMSXGroupSeq`.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value` in an internal representation. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `BYTE` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the `INT` datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the `long` datatype is 64 bits. Therefore, no range check is needed.Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a `long` property, then this function returns the value of the property. Otherwise it returns a null.

get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

get_string_property (property_name IN VARCHAR) RETURN VARCHAR
If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

Payload Methods

set_bytes (payload IN RAW)
Sets the internal payload as a RAW variable without any interpretation. The payload of `aq$_jms_map_message` is stored as either `RAW` or `BLOB` in the database. This member function sets a payload as a RAW variable without interpreting it.

set_bytes (payload IN BLOB)
Sets the internal payload as a BLOB variable without any interpretation. The payload of `aq$_jms_map_message` is stored as either `RAW` or `BLOB` in the database. This member function sets a payload as a BLOB variable without interpreting it.

get_bytes (payload OUT RAW)
Puts the internal payload into a `RAW` variable without any interpretation. The payload of `aq$_jms_map_message` is stored as either `RAW` or `BLOB` in the database. This member function gets a payload as raw bytes without interpreting it. Raises exceptions ORA-24190 if the length of internal payload is more than 32767.

get_bytes (payload OUT BLOB)
Puts the internal payload into a `BLOB` variable without any interpretation. The payload of `aq$_jms_map_message` is stored as either `RAW` or `BLOB` in the database. This member function gets a payload as a `BLOB` without interpreting it.

prepare (id IN PLS_INTEGER) RETURN PLS_INTEGER
Takes the byte array stored in `aq$_jms_map_message` and decodes it as a Java object in the Java stored procedure. The result of the decoding is stored as a static variable in Jserv session memory. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, then a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.
This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

flush (id IN PLS_INTEGER)
Takes the static variable in Jserv and synchronizes the content back to `aq$_jms_map_message`. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clear_body (id IN PLS_INTEGER)
Sets the Java stored procedure static variable to empty payload. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.
This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

clean (id IN PLS_INTEGER)
Closes and cleans up the `DataInputStream` or `DataOutputStream` at the Java stored procedure side corresponding to the operation ID. It is very important to call this procedure to avoid memory leaks. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clean_all
Closes and cleans up all the messages in the corresponding type of message store at the Java stored procedure side. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution.

set_boolean (id IN PLS_INTEGER, name IN VARCHAR2, value IN BOOLEAN)
Sets the Boolean value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_byte (id IN PLS_INTEGER, name IN VARCHAR2, value IN PLS_INTEGER)
Sets the `BYTE` value with the specified name in the map. Because there is no `BYTE` type in PL/SQL, `PLS_INTEGER` is used to represent a byte. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value IN RAW))
Sets the byte array value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value IN RAW, offset IN INT, length IN INT)
Sets a portion of the byte array value with the specified name in the map. Parameter `offset` is the initial offset within the byte array, and parameter `length` is the number of bytes to use. If the range [offset … offset+length] exceeds the boundary of the byte array value, then a Java

`IndexOutOfBounds` exception is thrown in the Java stored procedure and this procedure raises an ORA-24197 error. The index starts from 0. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value IN BLOB)
Sets the byte array value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value IN BLOB, offset IN INT, length IN INT)
Sets a portion of the byte array value with the specified name in the map. Parameter `offset` is the initial offset within the byte array, and parameter `length` is the number of bytes to use. If the range [offset … offset+length] exceeds the boundary of the byte array value, then a Java `IndexOutOfBounds` exception is thrown in the Java stored procedure, and this procedure raises an ORA-24197 error. The index starts from 0. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_char (id IN PLS_INTEGER, name IN VARCHAR2, value IN CHAR)
Sets the character value with the specified name in the map. If this value has multiple characters, then it is the first character that is used. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_double (id IN PLS_INTEGER, name IN VARCHAR2, value IN DOUBLE PRECISION)
Sets the double value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_float (id IN PLS_INTEGER, name IN VARCHAR2, value IN FLOAT)
This procedure is to set the float value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_int (id IN PLS_INTEGER, name IN VARCHAR2, value IN PLS_INTEGER)
Sets the int value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_long (id IN PLS_INTEGER, name IN VARCHAR2, value IN NUMBER)
Sets the long value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_short (id IN PLS_INTEGER, name IN VARCHAR2, value IN PLS_INTEGER)
Sets the short value with the specified name in the map. Because there is no short type in PL/SQL, `PLS_INTEGER` is used to represent a short. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_string (id IN PLS_INTEGER, name IN VARCHAR2, value IN VARCHAR2)
Sets the string value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

set_string (id IN PLS_INTEGER, name IN VARCHAR2, value IN CLOB))
Sets the string value with the specified name in the map. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

get_boolean (id IN PLS_INTEGER, name IN VARCHAR2) RETURN BOOLEAN
Retrieves the Boolean value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_byte (id IN PLS_INTEGER, name IN VARCHAR2) RETURN PLS_INTEGER
Retrieves the `BYTE` value with the specified name. If there is no item by this name, then null is returned. Because there is no `BYTE` type in PL/SQL, `PLS_INTEGER` is used to represent a byte. Although the PL/SQL users get an `PLS_INTEGER`, they are guaranteed that the value is in the Java `BYTE` value range. If this value is issued with a `set_byte` function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_bytes (id IN PLS_INTEGER, name IN VARCHAR2, value OUT NOCOPY BLOB)
Retrieves the byte array value with the specified name. If there is no item by this name, then null is returned. Because the size of the array might be larger than the limit of PL/SQL `RAW` type, a `BLOB` is always returned here. The `BLOB` returned is a copy, which means it can be modified without affecting the message payload. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_char (id IN PLS_INTEGER, name IN VARCHAR2) RETURN CHAR
Retrieves and returns the character value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid.

get_double (id IN PLS_INTEGER, name IN VARCHAR2) RETURN DOUBLE PRECISION
Retrieves and returns the double value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid.

get_float (id IN PLS_INTEGER, name IN VARCHAR2) RETURN FLOAT
Retrieves the float value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_int (id IN PLS_INTEGER, name IN VARCHAR2) RETURN PLS_INTEGER
Retrieves the `INT` value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_long (id IN PLS_INTEGER, name IN VARCHAR2) RETURN NUMBER
Retrieves the long value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_short (id IN PLS_INTEGER, name IN VARCHAR2) RETURN PLS_INTEGER
Retrieves the short value with the specified name. If there is no item by this name, then null is returned. Because there is no `short` type in PL/SQL, `INT` is used to represent a `short`. Although the PL/SQL users get an `PLS_INTEGER`, they are guaranteed that the value is in the Java short value range. If this value is issued with a `set_short` function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_string (id IN PLS_INTEGER, name IN VARCHAR2, value OUT NOCOPY CLOB)
Retrieves the string value with the specified name. If there is no item by this name, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

get_names (id IN PLS_INTEGER) RETURN aq$_jms_namearray
Retrieves all the names within the map message and returns them in a varray. Because `aq$_jms_namearray` has a size as 1024 and each element is a `VARCHAR(200)`, this function will return an error if the size of the name array of the payload exceeds the limit. Raises exception ORA-24195 if the size of the name array or the size of a name exceeds the limit.

get_names (id IN PLS_INTEGER, names OUT aq$_jms_namearray, offset IN PLS_INTEGER, length IN PLS_INTEGER) RETURN PLS_INTEGER
Retrieves a portion of the names within the map message. Because `aq$_jms_namearray` has a size as 1024 and each element is a `VARCHAR(200)`, this function will return an error if either limits are exceeded during the retrieval. (This means there is no sense to put a `length` parameter greater than 1024.) The index of the names of a map messages begins from 0. Parameter `offset` is the offset from which to start retrieving.
The function returns the number of names that have been retrieved. The names retrieved is the intersection of the interval [offset, offset+length-1] and interval [0, size-1] where size is the size of this map message. If the intersection is an empty set, then names will be returned as null and the function returns 0 as the number of names retrieved. If users iterate the names by retrieving in small steps, then this can be used to test that there are no more names to read from map message.
Raises exception ORA-24195 if the size of the name array or the size of a name exceed the limit, ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

get_object (id IN PLS_INTEGER, name IN VARCHAR2, value OUT NOCOPY AQ$_JMS_VALUE)
Returns a general value ADT `AQ$_JMS_VALUE`. If there is no item by this name, then null is returned.Users can use the `type` attribute of this ADT to interpret the data. See the map in the `AQ$_JMS_VALUE` ADT for the correspondence among `dbms_jms_plsql` package constants, Java datatype and `AQ$_JMS_VALUE` attribute. Note this member procedure might bring additional overhead compared to other `get` member procedures or functions. It is used only if the user does not know the datatype of the fields within a message before hand. Otherwise it is a good idea to use a specific `get` member procedure or function. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

get_size (id IN PLS_INTEGER) RETURN PLS_INTEGER
Retrieves the size of the map message. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

item_exists (id IN PLS_INTEGER, name IN VARCHAR2) RETURN BOOLEAN
Indicates that an item exists in this map message by returning `TRUE`. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

SYS.AQ$_JMS_STREAM_MESSAGE Type

This type is the ADT used to store a `StreamMessage` in an Oracle Database Advanced Queuing queue.

This section contains these topics:

- CONSTRUCT aq$_jms_stream_message Function
- JMS Header Methods
- System Properties Methods
- User Properties Methods
- Payload Methods

Syntax

```
TYPE aq$_jms_stream_message AS object(
 header     aq$_jms_header,
 bytes_len  int,
 bytes_raw  raw(2000),
 bytes_lob  blob,
 STATIC FUNCTION  construct RETURN aq$_jms_stream_message,
 MEMBER PROCEDURE set_replyto  (replyto IN sys.aq$_agent),
 MEMBER PROCEDURE set_type     (type    IN VARCHAR),
 MEMBER FUNCTION get_replyto  RETURN sys.aq$_agent,
 MEMBER FUNCTION get_type     RETURN VARCHAR,
 MEMBER PROCEDURE set_userid   (userid   IN VARCHAR),
 MEMBER PROCEDURE set_appid    (appid    IN VARCHAR),
 MEMBER PROCEDURE set_groupid  (groupid  IN VARCHAR),
 MEMBER PROCEDURE set_groupseq (groupseq IN INT),
 MEMBER FUNCTION get_userid   RETURN VARCHAR,
 MEMBER FUNCTION get_appid    RETURN VARCHAR,
 MEMBER FUNCTION get_groupid  RETURN VARCHAR,
 MEMBER FUNCTION get_groupseq RETURN INT,
 MEMBER PROCEDURE clear_properties,
 MEMBER PROCEDURE set_boolean_property(property_name IN VARCHAR,
   property_value IN BOOLEAN),
 MEMBER PROCEDURE set_byte_property   (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_double_property (property_name IN VARCHAR,
   property_value IN DOUBLE PRECISION),
 MEMBER PROCEDURE set_float_property  (property_name IN VARCHAR,
   property_value IN FLOAT),
 MEMBER PROCEDURE set_int_property    (property_name IN VARCHAR,
   property_value IN INT),
 MEMBER PROCEDURE set_long_property   (property_name IN VARCHAR,
   property_value IN NUMBER),
 MEMBER PROCEDURE set_short_property  (property_name IN VARCHAR,
   property_valuE IN INT),
 MEMBER PROCEDURE set_string_property (property_name IN VARCHAR,
   property_value IN VARCHAR),
 MEMBER FUNCTION get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN,
 MEMBER FUNCTION get_byte_property    (property_name IN VARCHAR) RETURN INT,
 MEMBER FUNCTION get_double_property  (property_name IN VARCHAR)
```

```
  RETURN DOUBLE PRECISION,
MEMBER FUNCTION get_float_property   (property_name IN VARCHAR) RETURN FLOAT,
MEMBER FUNCTION get_int_property     (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_long_property    (property_name IN VARCHAR) RETURN NUMBER,
MEMBER FUNCTION get_short_property   (property_name IN VARCHAR) RETURN INT,
MEMBER FUNCTION get_string_property  (property_name IN VARCHAR) RETURN VARCHAR,
MEMBER PROCEDURE set_bytes           (payload IN RAW),
MEMBER PROCEDURE set_bytes           (payload IN BLOB),
MEMBER PROCEDURE get_bytes           (payload OUT RAW),
MEMBER PROCEDURE get_bytes           (payload OUT BLOB),
MEMBER FUNCTION  prepare             (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER PROCEDURE reset               (id IN PLS_INTEGER),
MEMBER PROCEDURE flush               (id IN PLS_INTEGER),
MEMBER PROCEDURE clear_body          (id IN PLS_INTEGER),
MEMBER PROCEDURE clean               (id IN PLS_INTEGER),
STATIC PROCEDURE clean_all,
MEMBER FUNCTION  get_mode       (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_boolean   (id IN PLS_INTEGER) RETURN BOOLEAN,
MEMBER FUNCTION  read_byte      (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_bytes     (id IN PLS_INTEGER) RETURN BLOB,
MEMBER PROCEDURE read_bytes     (id IN PLS_INTEGER, value OUT NOCOPY BLOB),
MEMBER FUNCTION  read_char      (id IN PLS_INTEGER) RETURN CHAR,
MEMBER FUNCTION  read_double    (id IN PLS_INTEGER) RETURN DOUBLE PRECISION,
MEMBER FUNCTION  read_float     (id IN PLS_INTEGER) RETURN FLOAT,
MEMBER FUNCTION  read_int       (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_long      (id IN PLS_INTEGER) RETURN NUMBER,
MEMBER FUNCTION  read_short     (id IN PLS_INTEGER) RETURN PLS_INTEGER,
MEMBER FUNCTION  read_string RETURN CLOB,
MEMBER PROCEDURE read_string    (id IN PLS_INTEGER, value OUT NOCOPY CLOB),
MEMBER PROCEDURE read_object    (id IN PLS_INTEGER,
  value OUT NOCOPY AQ$_JMS_VALUE),
MEMBER PROCEDURE write_boolean  (id IN PLS_INTEGER, value IN BOOLEAN),
MEMBER PROCEDURE write_byte     (id IN PLS_INTEGER, value IN INT),
MEMBER PROCEDURE write_bytes    (id IN PLS_INTEGER, value IN RAW),
MEMBER PROCEDURE write_bytes    (id IN PLS_INTEGER, value IN RAW,
  offset IN INT, length IN INT),
MEMBER PROCEDURE write_bytes    (id IN PLS_INTEGER, value IN BLOB),
MEMBER PROCEDURE write_bytes    (id IN PLS_INTEGER, value IN BLOB,
  offset IN INT, length IN INT),
MEMBER PROCEDURE write_char     (id IN PLS_INTEGER, value IN CHAR),
MEMBER PROCEDURE write_double   (id IN PLS_INTEGER, value IN DOUBLE PRECISION),
MEMBER PROCEDURE write_float    (id IN PLS_INTEGER, value IN FLOAT),
MEMBER PROCEDURE write_int      (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_long     (id IN PLS_INTEGER, value IN NUMBER),
MEMBER PROCEDURE write_short    (id IN PLS_INTEGER, value IN PLS_INTEGER),
MEMBER PROCEDURE write_string   (id IN PLS_INTEGER, value IN VARCHAR2),
MEMBER PROCEDURE write_string   (id IN PLS_INTEGER, value IN CLOB));
```

CONSTRUCT aq$_jms_stream_message Function

STATIC FUNCTION construct RETURN aq$_jms_stream_message
Creates an empty `aq$_jms_stream_message` object.

JMS Header Methods

set_replyto (replyto IN sys.aq$_agent)
Sets the `replyto` parameter, which corresponds to `JMSReplyTo` in JMS.

set_type (type IN VARCHAR)
Sets the JMS type, which can be any text, and which corresponds to `JMSType` in JMS.

get_replyto RETURN sys.aq$_agent
Returns `replyto`, which corresponds to `JMSReplyTo`.

get_type RETURN VARCHAR
Returns `type`, which corresponds to `JMSType`.

System Properties Methods

set_userid (userid IN VARCHAR)
Sets `userid`, which corresponds to `JMSXUserID` in JMS.

set_appid (appid IN VARCHAR)
Sets `appid`, which corresponds to `JMSXAppID` in JMS.

set_groupid (groupid IN VARCHAR)
Sets `groupid`, which corresponds to `JMSXGroupID` in JMS.

set_groupseq (groupseq IN INT)
Sets `groupseq`, which corresponds to `JMSXGroupSeq` in JMS.

get_userid RETURN VARCHAR
Returns `userid`, which corresponds to `JMSXUserID`.

get_appid RETURN VARCHAR
Returns `appid`, which corresponds to `JMSXAppID`.

get_groupid RETURN VARCHAR
Returns `groupid`, which corresponds to `JMSXGroupID`.

get_groupseq RETURN NUMBER
Returns `groupseq`, which corresponds to `JMSXGroupSeq`.

User Properties Methods

clear_properties
Clears all user properties. This procedure does not affect system properties.

set_boolean_property (property_name IN VARCHAR, property_value IN BOOLEAN)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value` in an internal representation. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_byte_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -128 to 127 (8-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `BYTE` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_double_property (property_name IN VARCHAR, property_value IN DOUBLE PRECISION)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_float_property (property_name IN VARCHAR, property_value IN FLOAT)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_int_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -2147483648 to 2147483647 (32-bits). This check is necessary because in PL/SQL and Oracle Database, the `INT` datatype is 38 bits. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_long_property (property_name IN VARCHAR, property_value IN NUMBER)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. In PL/SQL and Oracle Database, the `NUMBER` datatype is 38 bits. In Java, the `long` datatype is 64 bits. Therefore, no range check is needed.Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

set_short_property (property_name IN VARCHAR, property_value IN INT)
Checks whether `property_name` is null or exists. If not, the procedure checks whether `property_value` is within -32768 to 32767 (16-bits). This check is necessary because neither PL/SQL nor RDBMS defines the `short` datatype. Raises exception ORA-24191 if the property name exists, ORA-24192 if the property name is null, or ORA-24193 if the property value exceeds the valid range.

set_string_property (property_name IN VARCHAR, property_value IN VARCHAR)
Checks whether `property_name` is null or exists. If not, the procedure stores `property_value`. Raises exception ORA-24191 if the property name exists or ORA-24192 if the property name is null.

get_boolean_property (property_name IN VARCHAR) RETURN BOOLEAN
If the property with the corresponding property name passed in exists, and if it is a `BOOLEAN` property, then this function returns the value of the property. Otherwise it returns a null.

get_byte_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `BYTE` property, then this function returns the value of the property. Otherwise it returns a null.

get_double_property (property_name IN VARCHAR) RETURN DOUBLE PRECISION
If the property with the corresponding property name passed in exists, and if it is a `DOUBLE` property, then this function returns the value of the property. Otherwise it returns a null.

get_float_property (property_name IN VARCHAR) RETURN FLOAT
If the property with the corresponding property name passed in exists, and if it is a `FLOAT` property, then this function returns the value of the property. Otherwise it returns a null.

get_int_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `Integer` property, then this function returns the value of the property. Otherwise it returns a null.

get_long_property (property_name IN VARCHAR) RETURN NUMBER
If the property with the corresponding property name passed in exists, and if it is a `long` property, then this function returns the value of the property. Otherwise it returns a null.

get_short_property (property_name IN VARCHAR) RETURN INT
If the property with the corresponding property name passed in exists, and if it is a `short` property, then this function returns the value of the property. Otherwise it returns a null.

get_string_property (property_name IN VARCHAR) RETURN VARCHAR
If the property with the corresponding property name passed in exists, and if it is a `STRING` property, then this function returns the value of the property. Otherwise it returns a null.

Payload Methods

get_bytes (payload OUT RAW)
Puts the internal payload into a RAW variable without any interpretation. The payload of type `aq$_jms_stream_message` is stored as either `RAW` or `BLOB` in the database. This member function gets a payload as raw bytes without interpreting it. Raises exception ORA-24190 if the length of internal payload is more than 32767.

get_bytes (payload OUT BLOB)
Puts the internal payload into a BLOB variable without any interpretation. The payload of type `aq$_jms_stream_message` is stored as either `RAW` or `BLOB` in the database. This member function gets a payload as a `BLOB` variable without interpreting it.

set_bytes (payload IN RAW)
Sets the internal payload as the RAW variable without any interpretation. The payload of type `aq$_jms_stream_message` is stored as either `RAW` or `BLOB` in the database. This member function sets a payload as raw bytes without interpreting it.

set_bytes (payload IN BLOB)
Sets the internal payload as the `BLOB` variable without any interpretation. The payload of type `aq$_jms_stream_message` is stored as either `RAW` or `BLOB` in the database. This member function sets a payload as a `BLOB` variable without interpreting it.

prepare (id IN PLS_INTEGER) RETURN PLS_INTEGER
Takes the byte array stored in `aq$_jms_stream_message` and decodes it as a Java object in the Java stored procedure. The result of the decoding is stored as a static variable in Jserv session memory. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, then a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID. This function also sets the message access mode to `MESSAGE_ACCESS_READONLY`. Subsequent calls of `write_XXX` procedure raise an ORA-24196 error. Users can call the `clear_body` procedure to set the message access mode to `MESSAGE_ACCESS_READONLY`.
This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

reset (id IN PLS_INTEGER)
Resets the starting position of the stream to the beginning and puts the stream message in `MESSAGE_ACCESS_READONLY` mode.

flush (id IN PLS_INTEGER)
Takes the static variable in Jserv and synchronizes the content back to `aq$_jms_stream_message`. This procedure will not affect the underlying access mode. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clear_body (id IN PLS_INTEGER)
Sets the Java stored procedure static variable to empty payload. Parameter `id` is used to identify the slot where the Java object is stored in the Oracle Database JVM session memory. If `id` is null, a new slot is created for this PL/SQL object. Subsequent JMS operations on the payload need to provide this operation ID.
It also sets the message access mode to `MESSAGE_ACCESS_WRITEONLY`. Later calls of `read_XXX` procedure raise ORA-24196 error. Users can call the `reset` or `prepare` procedures to set the message access mode to `MESSAGE_ACCESS_READONLY`. Write-only and read-only modes affect only the payload functions of `AQ$_JMS_BYTES_MESSAGE`. They do not affect the header functions.
This function raises ORA-24197 if the Java stored procedure throws an exception during execution, ORA-24198 if the operation ID is invalid, or ORA-24199 if the Java stored procedure message store overflows.

clean (id IN PLS_INTEGER)
Closes and cleans up the `DataInputStream` or `DataOutputStream` at the Java stored procedure side corresponding to the operation ID. It is very important to call this procedure to avoid memory leaks. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

clean_all
Closes and cleans up all the messages in the corresponding type of message store at the Java stored procedure side. This procedure raises ORA-24197 if the Java stored procedure throws an exception during execution.

get_mode (id IN PLS_INTEGER) RETURN PLS_INTEGER
Returns the current mode of this message. The return value is either `SYS.dbms_aqjms.READ_ONLY` or SYS.`dbms_aqjms.WRITE_ONLY`. Raises exception ORA-24197 if the Java stored procedure throws an exception during execution or ORA-24198 if the operation ID is invalid.

read_boolean (id IN PLS_INTEGER) RETURN BOOLEAN
Reads and returns a Boolean value from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_byte (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads and returns a byte value from the stream message. If the end of the message stream has been reached, then null is returned. Because there is no `BYTE` type in PL/SQL, `INT` is used to represent a byte. Although PL/SQL users get an `INT`, they are guaranteed that the value is in the Java `BYTE` value range. If this value is issued with a `write_byte` function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_bytes (id IN PLS_INTEGER) RETURN BLOB
Reads and returns a byte array from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid or ORA-24196 if the stream message is in write-only mode.

read_bytes (id IN PLS_INTEGER, value OUT NOCOPY BLOB)
Reads a byte array from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_char (id IN PLS_INTEGER) RETURN CHAR
Reads and returns a character value from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_double (id IN PLS_INTEGER) RETURN DOUBLE PRECISION
Reads and returns a double from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_float (id IN PLS_INTEGER) RETURN FLOAT
Reads and returns a float from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_int (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads and returns an `INT` from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_long (id IN PLS_INTEGER) RETURN NUMBER
Reads and returns a long from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_short (id IN PLS_INTEGER) RETURN PLS_INTEGER
Reads and returns a short value from the stream message. If the end of the message stream has been reached, then null is returned. Because there is no short type in PL/SQL, `PLS_INTEGER` is used to represent a `SHORT`. Although PL/SQL users get an `INT`, they are guaranteed that the value is in the Java short value range. If this value is issued with a `write_short` function, then there will not be an out of range error. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_string RETURN CLOB
Reads and returns a string from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion

between the type of real value and the expected type is invalid or ORA-24196 if the stream message is in write-only mode.

read_string (id IN PLS_INTEGER, value OUT NOCOPY CLOB)
Reads a string from the stream message. If the end of the message stream has been reached, then null is returned. Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

read_object (id IN PLS_INTEGER, value OUT NOCOPY AQ$_JMS_VALUE)
Returns a general value ADT `AQ$_JMS_VALUE`. Users can use the type attribute of this ADT to interpret the data. See Table 312-2 for the correspondence among `dbms_jms_plsql` package constants, Java datatype and `AQ$_JMS_VALUE` attribute. This member procedure might bring additional overhead compared to other read member procedures or functions. It is used only if the user does not know the datatype of the fields within a message beforehand. Otherwise it is a good idea to use a specific read member procedure or function.
Raises exception ORA-24194 if the type conversion between the type of real value and the expected type is invalid, ORA-24196 if the stream message is in write-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_boolean (id IN PLS_INTEGER, value IN BOOLEAN)
Writes a Boolean to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_byte (id IN PLS_INTEGER, value IN INT)
Writes a byte to the stream message. Because there is no `BYTE` type in PL/SQL, `INT` is used to represent a byte. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN RAW)
Writes a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN RAW, offset IN INT, length IN INT)
Writes a portion of a byte array as a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Parameter `offset` is the initial offset within the byte array, and parameter `length` is the number of bytes to use. If the range [offset, offset+length] exceeds the boundary of the byte array value, then a Java `IndexOutOfBounds` exception is thrown in the Java stored procedure. The index starts from 0.
Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN BLOB)
Writes a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_bytes (id IN PLS_INTEGER, value IN BLOB, offset IN INT, length IN INT)
Writes a portion of a byte array as a byte array field to the stream message. Consecutively written byte array fields are treated as two distinct fields when the fields are read. Parameter `offset` is the initial offset within the byte array, and parameter `length` is the number of bytes to use. If the range [offset, offset+length] exceeds the boundary of the byte array value, then a Java `IndexOutOfBounds` exception is thrown in the Java stored procedure. The index starts from 0.
Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_char (id IN PLS_INTEGER, value IN CHAR)
Writes a character value to the stream message. If this value has multiple characters, then it is the first character that is written. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_double (id IN PLS_INTEGER, value IN DOUBLE PRECISION)
Writes a double to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_float (id IN PLS_INTEGER, value IN FLOAT)
Writes a float to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_int (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes an `INT` to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_long (id IN PLS_INTEGER, value IN NUMBER)
Writes a long to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_short (id IN PLS_INTEGER, value IN PLS_INTEGER)
Writes a short to the stream message. Because there is no short type in PL/SQL, `INT` is used to represent a short. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_string (id IN PLS_INTEGER, value IN VARCHAR2)
Writes a string to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

write_string (id IN PLS_INTEGER, value IN CLOB)
Writes a string to the stream message. Raises exceptions ORA-24196 if the stream message is in read-only mode, ORA-24197 if the Java stored procedure throws an exception during execution, or ORA-24198 if the operation ID is invalid.

SYS.AQ$_JMS_OBJECT_MESSAGE Type

This type is the ADT used to store an `ObjectMessage` in an Oracle Database Advanced Queuing queue.

Syntax

```
TYPE aq$_jms_object_message AS object(
  header     aq$_jms_header,
  bytes_len  int,
  bytes_raw  raw(2000),
  bytes_lob  blob);
```

SYS.AQ$_JMS_NAMEARRAY Type

This type represents the name array returned by the `get_names` procedure of `aq$_jms_map_message`.

The maximum number of names this type can hold is 1024. The maximum length of each name is 200 characters.

Syntax

```
CREATE OR REPLACE TYPE AQ$_JMS_NAMEARRAY AS VARRAY(1024) OF VARCHAR(100);
```

Usage Notes

If the names array in the message payload is greater than 1024, then use the following function to retrieve the names in multiple portions:

```
MEMBER FUNCTION get_names(id IN PLS_INTEGER, names OUT aq$_jms_namearray,
   offset IN PLS_INTEGER, length IN PLS_INTEGER) RETURN PLS_INTEGER;
```

SYS.AQ$_JMS_VALUE Type

This type represents the general data returned by the `get_object` procedure of `aq$_jms_map_message` and the `read_object` procedure of `aq$_jms_stream_message`.

The `type` field in this ADT is used to decide which type of data this object is really holding. The following table lists the mapping between the `sys.dbms_jms_plsql` type constants, the corresponding Java type, and the data field of ADT `aq$_jms_value` which effectively holds the data.

Syntax

```
CREATE OR REPLACE TYPE AQ$_JMS_VALUE AS object(
 type      number(2),
 num_val   number,
 char_val  char(1),
 text_val  clob,
 bytes_val blob);
```

Table 312-2 AQ$_JMS_VALUE Type Fields and Java Fields

Type	Java Type	aq$_jms_value Data Field
`DBMS_JMS_PLSQL.DATA_TYPE_BYTE`	byte	num_val

Table 312-2 (Cont.) AQ$_JMS_VALUE Type Fields and Java Fields

Type	Java Type	aq$_jms_value Data Field
`DBMS_JMS_PLSQL.DATA_TYPE_SHORT`	short	num_val
`DBMS_JMS_PLSQL.DATA_TYPE_INTEGER`	int	num_val
`DBMS_JMS_PLSQL.DATA_TYPE_LONG`	long	num_val
`DBMS_JMS_PLSQL.DATA_TYPE_FLOAT`	float	num_val
`DBMS_JMS_PLSQL.DATA_TYPE_DOUBLE`	double	num_val
`DBMS_JMS_PLSQL.DATA_TYPE_BOOLEAN`	boolean	num_val: 0 FALSE, 1 TRUE
`DBMS_JMS_PLSQL.DATA_TYPE_CHARACTER`	char	char_val
`DBMS_JMS_PLSQL.DATA_TYPE_STRING`	java.lang.String	text_val
`DBMS_JMS_PLSQL.DATA_TYPE_BYTES`	byte[]	bytes_val

SYS.AQ$_JMS_EXCEPTION Type

This type represents a Java exception thrown on the Java stored procedure side.

The `id` field is reserved for future use. The `exp_name` stores the Java exception name, the `err_msg` field stores the Java exception error message, and the `stack` field stores the stack trace of the Java exception.

Syntax

```
CREATE OR REPLACE TYPE AQ$_JMS_EXCEPTION AS OBJECT (
   id           number, -- Reserved and not used. Right now always return 0.
   exp_name     varchar(200),
   err_msg      varchar(500),
   stack        varchar(4000));
```