Oracle® Communications MetaSolv Solution

EJB API Developer's Reference

Release 6.3

E69849-02

July 2017

ORACLE®

Oracle Communications MetaSolv Solution EJB API Developer's Reference, Release 6.3

E69849-02

Contents

Preface

This guide provides information on Oracle Communications MetaSolv Solution (MSS) Enterprise JavaBeans (EJB) set of APIs. This guide assists you to understand the details of implementing each MSS EJB API.

Audience

This guide is intended for developers who are developing applications that use the MetaSolv Solution EJB APIs. This guide assumes you have a working knowledge of the following:

- MSS
- EJB standards
- Java development

Related Documents

For more information, see the following documents in MSS 6.3 documentation set:

- *MSS Planning Guide:* Describes information you need to consider in planning your MSS environment prior to installation.
- *MSS System Administrator's Guide:* Describes postinstallation tasks and administrative tasks such as maintaining user security.
- *MSS Security Guide:* Provides guidelines and recommendations for setting up MSS in a secure configuration.
- *MSS Database Change Reference:* Provides information on the database changes in MSS releases.
- *MSS Network Grooming User's Guide:* Provides information about the MSS Network Grooming tool.
- *MSS Address Correction Utility User's Guide:* Provides information about the MSS Address Correction utility.
- *MSS Technology Module Guide:* Describes each of the MSS technology modules.
- *MSS Data Selection Tool How-to Guide:* Provides an overview of the Data Selection Tool, and procedures on how it used to migrate the product catalog, equipment specifications, and provisioning plans from one release of your environment to another.

- *MSS CORBA API Developer's Reference:* Describes how MSS APIs work, high-level information about each API, and instructions for using the APIs to perform specific tasks.
- *MSS Custom Extensions Developer's Reference:* Describes how to extend the MSS business logic with custom business logic through the use of custom extensions.
- *MSS Web Services Developer's Guide:* Describes the MSS Web Services and provides information about the MSS Web Service framework that supports web services, the various web services that are available, and how to migrate existing XML API interfaces to web service operations.

For step-by-step instructions for tasks you perform in MetaSolv Solution, log in to the application to see the online Help.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

1

Overview

This document provides information that you can use when working with the Oracle Communications MetaSolv Solution (MSS) Enterprise JavaBeans (EJB) application program interfaces (APIs).

About the MSS EJB APIs

This section provides an overview of EBJ APIs. You can use the EJB APIs to reconcile data from third-party applications into MSS.

Working with Connection EJB APIs

The "Working with Connections" chapter provides the set of EJB APIs that are related to connections, such as creating and updating connections, assigning and un-assigning ports, and also provisioning assignments. The following summarizes the APIs:

- **createConnectionsNew EJB API:** Creates multiple connections at a time based on the input data. See "Creating Connections (createConnectionsNew)" for more information.
- **associateConnectionToNetworkSystem EJB API:** Associates the given connection between two components of a network system or deletes the association between components for multiple connections at a time in a network system. See "Associating/Unassociating Connections (associateConnectionToNetworkSystem)" for more information.
- **autoBuildConnection EJB API:** Auto-builds an optical circuit to lower levels based on the input data and hierarchy (service type) structure defined in MSS, including simultaneous auto-build of multiple connections that are a part of network systems. See "Auto-Building Connections (autoBuildConnection)" for more information.
- **assignPort EJB API:** Creates new port assignments and removes the existing port assignments for multiple connections at a time. See "Assigning/Unassigning Ports (assignPort)" for more information.
- **createCustomerConnection EJB API:** Creates multiple customer connections at a time based on the input data. See "Creating Customer Connections (createCustomerConnection)" for more information.
- **updateProvisioningInfo EJB API:** Enables you to provision:
  - Network assignment (optical/SDH provisioning)
  - Facility assignment
  - Equipment assignment

See "Provisioning Assignments for Customer Circuits (updateProvisioningInfo)" for more information.

- **createCustomerConnNew EJB API:** Creates and provisions multiple customer connections at a time based on the input data. See "Creating and Provisioning Customer Connections (createCustomerConnNew)" for more information.
- **createVirtualConnection EJB API:** Creates and provisions customer/non-customer virtual connections based on the input data. See "Creating Virtual Connections (createVirtualConnection)" for more information.
- **provisionVirtualConnection EJB API:** Updates the provisioning information for non-channelized or virtual connections. See "Provisioning Virtual Connections (provisionVirtualConnection)" for more information.
- **updateConnection EJB API:** Updates multiple connections. Each connection is updated based on the input data. See "Updating Connections (updateConnection)" for more information.

Working with Engineering Work Orders

The "Working with Engineering Work Orders" chapter provides the set of EJB APIs that are related to Engineering Work Orders, creating and updating work orders and their notes, associating connections and equipment to work orders, and also processing due date supplements. The following summarizes the EJB APIs:

- **createWorkOrder EJB API:** Creates an Engineering Work Order. The new order is created based on the input data. See "Creating a Work Order (createWorkOrder)" for more information.
- **updateWorkOrder EJB API:** Updates an Engineering Work Order. The order is updated based on the input data. See "Updating a Work Order (updateWorkOrder)" for more information.
- **createWorkOrderNote EJB API:** Creates a note for an Engineering Work Order. See "Creating a Work Order Note (createWorkOrderNote)" for more information.
- **updateWorkOrderNote EJB API:** Updates a note for an Engineering Work Order. See "Updating a Work Order Notes (updateWorkOrderNote)" for more information.
- **associateConnectionToWorkOrder EJB API:** Associates a connection to an Engineering Work Order. See "Associating a Connection to a Work Order (associateConnectionToWorkOrder)" for more information.
- **associateEquipmentToWorkOrder EJB API:** Associates equipment to an Engineering Work Order. See "Associating Equipment to a Work Order (associateEquipmentToWorkOrder)" for more information.
- **addTask EJB API:** Assigns a task to an Engineering Work Order. See "Adding a Task to a Work Order (addTask)" for more information.
- **processDDChangeSupplement EJB API:** Processes the supplementing of an Engineering Work Order. See "Supplementing a Work Order (processDDChangeSupplement)" for more information.

Overview of EJB API Content

This document includes the following:

- Information about each MSS EJB API, which describes API usage patterns for implementing common business scenarios.
- Examples that show correct usage setting parameters for the APIs.
- Input information for an API.
- Output information received from an API, including the following:
  - Data that has reconciled successfully into MSS.
  - Data that failed to reconcile with MSS. In this case, all the validation errors are displayed based on the input provided.
- Working with transactions. A transaction provides information about:
  - Whether or not a specific API participates in the client's transaction
  - How the input data is reconciled by the API

Calling an API

This section provides information about the steps that are required to call an API.

Calling an API typically includes:

- Getting the initial context
- Looking up the context with the Java Naming and Directory Interface (JNDI) name
- Calling the specific API on the business interface reference
- Getting the results from the API output.
- Getting the error codes/messages from the API.

> **Note:** The MSS application implements security for EJB methods. You must add a registered user to the Global Role MSSRole to access the EJB methods externally. See *MSS Security Guide* for more information.

For each API the parameters are listed and indicate whether each is mandatory or optional. If you do not provide a mandatory parameter, the API fails and returns an error. For all parameters, if you provide a value that does not exist in MSS for a parameter where a valid value or identifier is applicable, the API fails and returns an error message.

Example 1–1 shows the information required to call an API.

Example 1–1 Sample Code for an API Call

```
// Gets the initial context.
// Context.INITIAL_CONTEXT_FACTORY is a constant that holds the initial context
// factory to use.
// Context.PROVIDER_URL is a constant that holds the information for the service
// provider to use.
// IP_Address is the IP address of the system where the WebLogic server is
// running for MSS.
// WebLogic_Domain_Port is the port of the server on which the WebLogic server is
// running.
// Context.SECURITY_PRINCIPAL is a constant that holds the user name for
// authenticating to the service.
```

```
// Context.SECURITY_CREDENTIALS is a constant that holds the password of the user
// for authenticating to the service.
Properties h = new Properties();
h.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFactory");
h.put(Context.PROVIDER_URL, "t3://IP_Address:WebLogic_Domain_Port");
h.put(Context.SECURITY_PRINCIPAL, "LoginId");
h.put(Context.SECURITY_CREDENTIALS, "password");
Context ctx = new InitialContext(h);

// Looks up the context with the JNDI name and gets the reference of
// ConnectionAccessManagerRemote business interface object.
ConnectionAccessManagerRemote remote = (ConnectionAccessManagerRemote) =
ctx.lookup("nrm/resource/entity/connection/ConnectionAccessManager");

// Calls the corresponding API on the remote object to import the required
// data.
// API_CALL can be, for example, createVirtualConnection(Connection[]
// conns, User user).
remote.API_CALL

// Gets the results from the remote object reference.
results.getReturnObject();
// The resultant must be converted to the corresponding object type based on the
// calling API. For example:
ArrayList keys =
        (ArrayList)results.getReturnObject();
          for (int i = 0; i < keys.size(); i++) {
            System.out.println(keys.get(i));
          }

// Gets the error messages from the API.
results.getMessages();
// Always returns a vector of the results. For example:
Vector errorMessages = (Vector)results.getMessages();
         Iterator errorIterator = errorMessages.iterator();
          while (errorIterator.hasNext()) {
            MSLVException mslvExcep = (MSLVException)errorIterator.next();
            System.out.println("Error code: "+ mslvExcep.getCode() + ", Error
                        message : " + mslvExcep.getMessage());
          }
```

2

Working with Connections

This chapter provides information about the API methods that have to do with connections, such as creating connections, associating connections, assigning ports and auto-building connections.

Creating Connections (createConnectionsNew)

This API supports multiple connections at a time and creates a connection based on the input data. If customer attribute (CA) data is specified in the input, this API populates the CAs for connection. Otherwise, it just creates a connection. You can also use the other newly added APIs along with this API. Using this API, association of a connection to a network system, port assignments for connection, and auto-build can be done at the time of creating the circuit. To accomplish this, you must provide the input data relevant to each API. If connection name is not provided in the input, it would be auto-generated based on the input flag value (autoEcckt). If the flag is enabled, the ECCKT is auto-generated. In this case, the ECCKT (connection name) generation is done based on the ECCKT type of the connection. This API does not support trunks and telephone number format circuit creation.

EJB API Call

```
ConnectionAccessManagerRemote.createConnectionsNew(
   Connection[] connections,
   User user)
```

Container Object

```
com.metasolv.value.resource.entity.connection.Connection[]
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns an array list of successfully processed connections.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 2–1 lists the input parameters for the createConnectionsNew API.

Table 2–1 Input Parameters for the API

Input Parameter	Parameter Information
connections	**Data Type:** com.metasolv.value.resource.entity.connection.Connection[] **Description:** Contains information to run the functionality for creating connections. This array can contain input of connections. For example, if you want to create ten connections as part of this API call, you must create an array with all the ten connection objects and pass it to the API. For example, suppose that for the third connection, the API creates the connection, adds labels and values, and populates custom attributes. However, if the input provisioning information is incorrect, the API fails and does not provision the connection. In this case, the following occur for the third connection: ■ The connection is created ■ The changes related to the creation of labels and values/custom attributes for the connection are committed ■ The changes related to the provisioning of the connection are rolled back ■ The API commits the changes related to the first, second, and third connections and continues to process the fourth connection, then the fifth connection, and so on If during the API processing, the creation of the third and sixth connections fails, the results object (which you get from the API return object) contains the errors related only to the third and sixth connections. The data related to all the remaining connections is imported/committed into the MSS database. This parameter is mandatory.
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.

Table 2–2 lists the containers you must set within the Connection container object.

Table 2–2 Containers To Set Within the Connection Container Object

Input Parameter	Data Type
caDataContainer[]	com.metasolv.value.resource.CaContainer
portContainer	com.metasolv.value.resource.PortAssignmentContainer
autoBuildContainer	com.metasolv.value.resource.AutoBuildConnectionContainer
networkAssociationContainer	com.metasolv.value.resource.ConnectionNSAssignmentContainer

Table 2–3 lists the input parameters that you set in the Connection object.

Table 2–3 Connection Input Parameters

Input Parameter	Parameter Information
eccktType	**Data Type:** com.metasolv.value.resource.entity.connection.EccktType **Description:** Indicates the type of circuit identification assigned to this circuit. Valid values are: ■ **CLF** (Common Language Facility) ■ **CLS** (Common Language Serial) ■ **OTF** (Free format of the CLF - unformatted facility) ■ **OTS** (Free format of the CLS - unformatted serial) This parameter is mandatory.
ratecode	**Data Type:** com.metasolv.value.resource.RateCode **Description:** Indicates the rate code of the connection. You must specify a value for the rate code. This parameter is mandatory.
serviceType	**Data Type:** com.metasolv.value.service.ServiceType **Description:** You must set both the service type category and service type code in this object. This parameter is mandatory.
exchangeCarrierId	**Data Type:** String **Description:** Connection identifier. If connection identifier (connection name) must be auto-generated, this is optional; else it is mandatory.
connectionType	**Data Type:** com.metasolv.value.resource.entity.connection.ConnectionType **Description:** Indicates the type of the circuit. Valid values are: ■ **F** (Facility) ■ **S** (Special) ■ **B** (Bandwidth) ■ **C** (NGN) This parameter is mandatory.
connectionStatus	**Data Type:** com.metasolv.value.resource.entity.connection.ConnectionStatus **Description:** Describes the current operational status of the connection. Valid values are: ■ **1** = Pending ■ **2** = Assigned ■ **3** = In Progress ■ **4** = CLR Issued ■ **5** = DLR Issued ■ **6** = In Service ■ **7** = Pending Disconnect ■ **8** = Disconnected ■ **9** = Problem ■ **A** = Cancelled This parameter is mandatory.

Table 2–3 (Cont.) Connection Input Parameters

Input Parameter	Parameter Information
ALocation	**Data Type:** com.metasolv.value.location.Location **Description:** Originating location of the connection. You must set either the location ID (integer) or the location short name (string) in this object. This parameter is mandatory.
ZLocation	**Data Type:** com.metasolv.value.location.Location **Description:** Terminating location of the connection. You must set either the location ID (integer) or the location short name (string) in this object. This parameter is mandatory.
jurisdictionCode	**Data Type:** com.metasolv.value.resource.JurisdictionCode **Description:** A code classifying the circuit related to its originating/terminating end for toll separation purposes. Valid values are: ■ **1** = Access Revenue - InterState/InterLATA ■ **2** = Access Revenue - IntraState/InterLATA ■ **3** = Access Revenue - InterState/IntraLATA ■ **4** = Access Revenue - IntraState/IntraLATA ■ **5** = Local Revenue - Exchange (Bill & Keep) ■ **6** = Local Revenue - Interexchange (Bill & Keep) ■ **7** = Survivable Host Remote ■ **8** = Mux & Non Survivable ■ **X** = Miscellaneous Carrier ■ **Y** = Ltr Facility ■ **0** = N/A (Multiple) This parameter is optional.
createConnectionPositions	**Data Type:** boolean **Description:** If set to **true**, it creates the default connection positions. If set to **false**, the connection positions are not created. By default, it is set to **false**. Applicable to facility connections. This parameter is mandatory depending on the scenario in which it is used.
autoEcckt	**Data Type:** boolean **Description:** To generate the connection name, this flag should be set to **true**. If the connection name is not provided and this flag is **true**, the ECCKT is auto-generated according to the format. By default, the autoEcckt is set to **false**. This parameter is mandatory depending on the scenario in which it is used.
designator	**Data Type:** String **Description:** Represents the facility designation of the connection. Required only when you want to create a connection with a particular CLF designation. If it is not given in the input, the designation is auto-generated. This parameter is optional.

Table 2–3 (Cont.) Connection Input Parameters

Input Parameter	Parameter Information
telcoId	**Data Type:** String **Description:** Used for CLS-format ECCKT. This parameter is mandatory depending on the scenario in which it is used. Applicable only for the CLS-format connections to get auto-generated ECCKT.
eccktModifier	**Data Type:** String **Description:** Used for CLS-format ECCKT. This parameter is mandatory depending on the scenario in which it is used. Applicable only for the CLS-format connections to get auto-generated ECCKT.
eccktPrefix	**Data Type:** String **Description:** You can specify this parameter if the autoEcckt flag is set to **true** and the circuit format is CLS or CLT. This parameter is optional.
eccktSuffix	**Data Type:** String **Description:** You can specify this parameter if the autoEcckt flag is set to **true** and the circuit format is CLS. This parameter is optional.
eccktSegment	**Data Type:** String **Description:** Used for CLS-format ECCKT. This parameter is mandatory depending on the scenario in which it is used. Applicable only for the CLS-format connections to get auto-generated ECCKT.
partitionGroupName	**Data Type:** String **Description:** Name of the partition group. This parameter is mandatory if the partition group is selected in the **Active Partition Groups** preference.
conSpecificationName	**Data Type:** String **Description:** The name of the connection specification. This parameter is mandatory only if CaContainer is populated.
caDataContainer[]	**Data Type:** com.metasolv.value.resource.CaContainer **Description:** List of custom attributes (CA) name/value pairs. This data is required only when CA data must be populated for the new connection. This parameter is mandatory depending on the scenario in which it is used. To populate custom attributes (CA) for a connection, this container should be populated. See Table 2–32, " CaContainer Input Parameters" for more information.

Table 2–3 (Cont.) Connection Input Parameters

Input Parameter	Parameter Information
portContainer	**Data Type:** com.metasolv.value.resource.PortAssignmentContainer **Description:** Required if the port assignments must be made for a connection after it is created. You must populate this container to assign a port for connection. This parameter is mandatory depending on the scenario in which it is used. See Table 2–9, " PortAssignmentContainer Input Parameters" for more information.
autoBuildContainer	**Data Type:** com.metasolv.value.resource.AutoBuildConnectionContainer **Description:** Required if the connection must be auto-built to lower channel levels. You must populate this container to auto-build a connection. This parameter is mandatory depending on the scenario in which it is used. See Table 2–7, " AutoBuildConnectionContainer Input Parameters" for more information.
networkAssociationContainer	**Data Type:** com.metasolv.value.resource.ConnectionNSAssignmentContainer **Description:** Required only when the connection that has been created must be associated with the network system. You must populate this container to associate a connection with a network system. This parameter is mandatory depending on the scenario in which it is used. See Table 2–5, " ConnectionNSAssignmentContainer Input Parameters" for more information.

Sample Data for Input Parameters

Example 2–1 is an example of the data values you must set for the input parameters to create a CLF type connection. The example also includes information to auto-generate the ECCKT.

Example 2–1 Data Values for Input Parameters

```
//Specify the type of circuit identification you want to assign to the connection.
clsType =
validValueAccessManager.getValue(com.metasolv.value.resource.entity.connection.Ecc
ktType.TYPE, "CLF");

//Specify the rate code for the connection.
com.metasolv.value.resource.RateCode rc =
com.metasolv.value.resource.RateCode)valueFact.makeInstance(com.metasolv.value.res
ource.RateCode.class);
rc.setCode("OC12");

//Specify the service type for the connection. You must set both the service type
//category and service type code in this object.
com.metasolv.value.service.ServiceType
st=(com.metasolv.value.service.ServiceType)valueFact.makeInstance(com.metasolv.val
ue.service.ServiceType.class);
st.setServiceTypeCode("OC12");
st.setServiceTypeCategory("CLFI");

//Specify the type of circuit you want to create.
com.metasolv.value.resource.entity.connection.ConnectionType
ct=validValueAccessManager.getValue(com.metasolv.value.resource.entity.connection.
ConnectionType.TYPE, 'F');

//Specify the current operational status of the connection.
```

```
com.metasolv.value.resource.entity.connection.ConnectionStatus cs =
validValueAccessManager.getValue(com.metasolv.value.resource.entity.connection.Con
nectionStatus.TYPE,'6');

//Specify the originating location of the connection.
com.metasolv.value.location.Location
Aloc=(com.metasolv.value.location.Location)valueFact.makeInstance(com.metasolv.val
ue.location.Location.class); Aloc.setLocationCode("CSVLILXE");

//Specify the terminating location of the connection.
com.metasolv.value.location.LocationZloc=(com.metasolv.value.location.Location)val
ueFact.makeInstance(com.metasolv.value.location.Location.class);
Zloc.setLocationId(91);

// Set the required information to auto-generate ECCKT.
// <prefix><service type><mod><serial number><suffix><Telco id><segment>
conn.setAutoEcckt(true);

//Set the required information to create the default connection positions.
connection.setCreateConnectionPositions(true)
```

Note: A new connection 103/OC12/CSVLILXE/NBRKILNT52T will be created based on the specified data values.

Associating/Unassociating Connections (associateConnectionToNetworkSystem)

This API associates the given connection between two components of a network system or deletes the association between components in a network system. The API associates or unassociates the connection based on the setting of the associationFlag parameter in the ConnectionNSAssignmentContainer. This API does the association if this flag is set to **true**; else, it does the unassociation. In case of unassociation, this API deletes the old association. To make a new network association, you must invoke the API with relevant data of network association. This API supports association/unassociation of multiple connections at a time.

Note: This API does not support SONET network systems.

EJB API Call

```
ConnectionAccessManagerRemote.associateConnectionToNetworkSystem(
                ConnectionNSAssignmentContainer[] containers,
                User user)
```

Container Object

```
com.metasolv.value.resource.ConnectionNSAssignmentContainer[]
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns an array list of connections that have been successfully processed.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 2–4 shows the input parameters for the API.

Table 2–4 Input Parameters for the API

Input Parameter	Parameter Information
containers	**Data Type:** com.metasolv.value.resource.ConnectionNSAssignmentContainer[] **Description:** Contains information to run the functionality to associate/unassociate connections to network systems. This array can contain input for connections. If you want to associate/unassociate ten connections to network systems as part of this API call, you must create an array with all the ten connection objects and pass it to the API. For example, if the third connection is not associated to a network system due to any incorrect input data, the API commits the data related to the first and second connections and rollbacks the changes for only the third connection and continues to process the fourth connection, then the fifth connection, and so on. This parameter is mandatory.
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.

Table 2–5 lists the input parameters that you set in the ConnectionNSAssignmentContainer object.

Table 2–5 ConnectionNSAssignmentContainer Input Parameters

Input Parameter	Parameter Information
connectionIdentifier	**Data Type:** String **Description:** Connection ECCKT or identifier to uniquely differentiate the connection. Represents the connection that must be associated with the network system. This parameter is mandatory.
nsName	**Data Type:** String **Description:** Network system name with which the input connection must be associated. This parameter is mandatory.
parentCompName	**Data Type:** String **Description:** Component name at A location. This parameter is mandatory.
childCompName	**Data Type:** String **Description:** Component name at Z location. The given connection would be associated between component A and component Z. This parameter is mandatory.
parentCompNbr	**Data Type:** Int **Description:** Used along with the component name to identify the component. Useful only when multiple components with the same name exist. This parameter is mandatory if the component name contains a number.

Table 2–5 (Cont.) ConnectionNSAssignmentContainer Input Parameters

Input Parameter	Parameter Information
childCompNbr	**Data Type:** Int **Description:** Used along with the component name to identify the component. Useful only when multiple components with the same name exist. This parameter is mandatory if the component name contains a number.
parentCompNsName	**Data Type:** String **Description:** Represents the network system name of the parent component. It is required when the network is embedded in the other network system. This field must be populated if the component is part of the embedded network system, which is different from the outer network.
childCompNsName	**Data Type:** String **Description:** Represents the network system name of the child component. It is required if the child network is embedded in the other network system. This field must be populated if the component is part of the embedded network system, which is different from the outer network.
connectionSpecName	**Data Type:** String **Description:** Connection specification name with which the connection must be associated. This parameter is mandatory.
associationFlag	**Data Type:** boolean **Description:** A boolean field to determine the association/unassociation operation. The **true** and **false** values represent association and unassociation operations respectively. By default this field is set to **false**. To associate a connection, you must set this flag to **true**. To unassociate a connection, you must set this flag to **false**.

Sample Data for Input Parameters

Example 2–2 is an example of the data values you must set for the input parameters to associate connections with network systems. In this example, the connection would be associated between **Testcomp1** and **Testcomp2** in the network system **Test Main.** Here, both the end components are part of the embedded networks in **Test Main**. Hence, embedded network names **Embednetwork1** and **Embednetwork2** should be provided in the input.

Example 2–2 Data Values for Input Parameters

```
//Create the connection assignment input instance.
ConnectionNSAssignmentContainer cnsac = new ConnectionNSAssignmentContainer();
//Specify the connection ECCKT or identifier that must be associated with
//the network system.
cnsac.setConnectionIdentifier("ASC18/OC12/AAAA1112/BBCYCAXF");

//Specify the connection specification name with which you want
//to associate the connection.
cnsac.setsetConnectionSpecName("Facility Circuit");
//Specify the network system name with which the input connection
//must be associated.
cnsac.setNsName("Test Main");
//Specify the parent component name at the originating (A) location.
cnsac.setParentCompName("Testcomp1");
```

```
//Specify the child component name at the terminating (B) location.
cnsac.setChildCompName("Testcomp2");
//Specify the parent component number, which is used along with the component
//name to identify the component.
cnsac.setParentCompNbr(1);
//Specify the child component number, which is used along with the component
//name to identify the component.
cnsac.setChildCompNbr(1);
//Specify the network system name of the parent component. This parameter
//is required when the network is embedded in the other network system.
cnsac.setParentCompNsName("Embednetwork1");
//Specify the network system name of the child component. This parameter
//is required when the network is embedded in the other network system.
cnsac.setChildCompNsName("Embednetwork2");
//Set the association flag, which determines the association/unassociation
//operation. To associate a connection, you must set this flag to true. To
//unassociate a connection, you must set this flag to false.
cnsac.setAssociationFlag(true);
```

Auto-Building Connections (autoBuildConnection)

This API autobuilds an optical circuit to lower levels based on the input data and hierarchy (service type) structure defined in MSS. This API supports simultaneous autobuild of multiple connections. This API supports only the autobuild of connections that are a part of network systems.

Note: This API is valid only for optical network systems.

EJB API Call

```
ConnectionAccessManagerRemote.autoBuildConnection(
   AutoBuildConnectionContainer[] containers,
   User user)
```

Container Object

```
com.metasolv.value.resource.AutoBuildConnectionContainer[]
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns an array list of successfully processed connections.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 2–6 shows the input parameters for the API.

Table 2–6 Input Parameters for the API

Input Parameter	Parameter Information
containers	**Data Type:** com.metasolv.value.resource.AutoBuildConnectionContainer[] **Description:** Contains information to run the functionality to autobuild connections. This array can contain input for connections. If you want to autobuild ten virtual connections as part of this API call, you must create an array with all the ten connection objects and pass it to the API. For example, if the autobuild for the third connection fails due to any incorrect input data, the API commits the data related to the first and second connections and rollbacks the changes for only the third connection and continues to autobuild the fourth connection, then the fifth connection, and so on. This parameter is mandatory.
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.

Table 2–7 lists the input parameters that you set in the AutoBuildConnectionContainer object.

Table 2–7 AutoBuildConnectionContainer Input Parameters

Input Parameter	Parameter Information
connectionIdentifier	**Data Type:** String **Description:** Connection ECCKT or identifier to uniquely differentiate the connection. Represents the connection that must be autobuilt to lower channel levels. This parameter is mandatory.
riddenRateCodes	**Data Type:** TreeMap **Description:** Rate code of the positions that you want to autobuild. Input should be specified as a TreeMap structure, which should have the key as the autobuild level and the rate code as the value. This parameter is mandatory.

Table 2–7 (Cont.) AutoBuildConnectionContainer Input Parameters

Input Parameter	Parameter Information
riderRateCodes	**Data Type:** TreeMap **Description:** Rate code to which the available circuit positions are to be built. Input should be specified as a TreeMap structure, which should have the key as the autobuild level and the rate code as the value. This parameter is mandatory.
numberOfChannelsToBuild	**Data Type:** TreeMap **Description:** Number of channel positions to be autobuilt using the ridden rate codes and the rider rate codes. Input should be specified as a TreeMap structure, which should have the key as the autobuild level and the number of channel positions as the value. This parameter is optional. Use this parameter to provide information about the selective channels to be autobuilt. If this is not populated, the first available channels are built in top-down approach.
channelNumbersToBuild	**Data Type:** TreeMap **Description:** The channel numbers to be autobuilt using the ridden rate codes and the rider rate codes. Input should be specified as a TreeMap structure, which should have the key as the autobuild level and a vector of string arrays containing the channel numbers to be autobuilt as the value. This parameter is optional. Use this parameter to provide information about the selective channels to be autobuilt. If this is not populated, the first available channels are built in top-down approach.

Sample Data for Input Parameters

Example 2–3 is an example of the data values you must set for the input parameters to auto-build an OC12 connection to two levels.

Example 2–3 Data Values for Input Parameters

```
//Create the update connection input instance.
AutoBuildConnectionContainer abcc = new AutoBuildConnectionContainer();

//Specify the connection ECCKT or identifier for the connection that must be
autobuilt to lower channel levels.
abcc.setConnectionIdentifier("898/OC12/LocA/LocZ");

//Specify the rate code of the positions that you want to autobuild.
TreeMap riddenRc = new TreeMap();
riddenRc.put("1","STS1"); riddenRC.put("2","VT1");

//Specify the rate code to which the available circuit positions are to be built.
TreeMap riderRc = new TreeMap();
riderRc.put("1","VT1"); riderRc.put("2","DS1");

//Specify the number of channel poisitions to be autobuilt using the ridden rate
//codes and the rider rate codes.
TreeMap numberOfChannelsToBuild = new TreeMap();
numberOfChannelsToBuild.put("1","12");
numberOfChannelsToBuild.put("2","336");
```

```
//Specify the channel numbers to be autobuilt using the ridden rate codes and the
//rider rate codes.
//For level 1: To build channels 2 and 5 at STS1 positions.
TreeMap channelNumbersToBuild = new TreeMap();
String [] chans = {"2","5"}; Vector v1 = new Vector(); v1.add(chans);
channelNumbersToBuild.put("1",v1);
//For level 2: To build channels 4 and 6 under 2 at VT1 positions.
String [] chans11 = {"2","4"}; Vector v2 = new Vector(); v2.add(chans11);
String [] chans12 = {"2","6"}; Vector v2 = new Vector(); v2.add(chans12);
//For level 2: To build channels 3 and 9 under 5 at VT1 positions.
String [] chans21 = {"5","3"}; Vector v2 = new Vector(); v2.add(chans21);
String [] chans22 = {"5","9"}; Vector v2 = new Vector(); v2.add(chans22);
channelNumbersToBuild: channelNumbersToBuild.put("2",v2);
```

Note: These parameters can be extended for the next levels based on the requirement.

Assigning/Unassigning Ports (assignPort)

You can use this API to assign or unassign port assignments. This API supports port assign/unassign operations of multiple connections at a time. You must set the assign parameter in the PortAssignmentContainer to true for assigning ports. To make new assignments, you must invoke this API using the relevant data of port assignment.

EJB API Call

```
ConnectionAccessManagerRemote.assignPort(PortAssignmentContainer[] containers,
                                         User user)
```

Container Object

```
com.metasolv.value.resource.PortAssignmentContainer[]
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns an array list of successfully processed connections.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 2–8 shows the input parameters for the API.

Table 2–8 Input Parameters for the API

Input Parameter	Parameter Information
containers	**Data Type:** com.metasolv.value.resource.PortAssignmentContainer[] **Description:** Contains information to run the functionality to associate/unassociate ports for connections. This array can contain input for connections. If you want to associate/unassociate ports for ten connections to network systems as part of this API call, you must create an array with all the ten connection objects and pass it to the API. For example, if the port assignment for the third connection fails due to any incorrect input data, the API commits the data related to the first and second connections and rollbacks the changes for only the third connection and continues to process the fourth connection, then the fifth connection, and so on. This parameter is mandatory.
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.

Table 2–9 lists the input parameters that you set in the PortAssignmentContainer object.

Table 2–9 PortAssignmentContainer Input Parameters

Input Parameter	Parameter Information
aZOtherCd	**Data Type:** String **Description:** Indicates the location where you must make the assignment. Valid values are: ■ **A** = A location ■ **Z** = Z location ■ **O** = On the circuit but not at the A or Z location This parameter is optional.
assign	**Data Type:** boolean **Description:** Indicates whether this is a port assignment (true) or unassignment (false). To assign a port, set this field to **true**. To unassign a port, set this field to **false**. This parameter is mandatory depending on the scenario in which it is used.
rootEquipmentId	**Data Type:** Long **Description:** Root (base) equipment ID (rack). This parameter is mandatory.

Table 2–9 (Cont.) PortAssignmentContainer Input Parameters

Input Parameter	Parameter Information
mountingPositions	**Data Type:** ArrayList **Description:** An array of integers that represents mounting positions of the equipment hierarchy. If the final equipment to which the connection must be assigned is mounted on fourth position of the shelf and the shelf, in turn, is mounted on the second mounting position of the rack, then the object must be set to values **2** and **4**. This parameter is optional.
circuitIdentifier	**Data Type:** String **Description:** Connection ECCKT or identifier to uniquely differentiate the connection. Represents the connection that must be assigned/unassigned. This parameter is mandatory.
portSeq	**Data Type:** Long **Description:** Port number of the equipment where the circuit must be assigned. This parameter is mandatory.

Sample Data for Input Parameters

Example 2–4 is an example of the data values you must set for the input parameters to assign ports for connections. In this example, the input connection will be assigned to the sixth port of the card. The card hierarchy is: rack (second position) - shelf (fourth position) - card.

Example 2–4 Data Values for Input Parameters

```
//Specify the location where you want the assignment. For example, A, Z, or O.
container.setAZOtherCd("A");

//Specify whether you want to assign or unassign ports for connections. To assign
//a port, set this parameter to true. To unassign a port, set this to false.
container.setAssign(true);

//rootEquipmentID specifies the base equipment on which the final card is mounted.
//Integer Mp1 represents a shelf on the mounting position of the base equipment;
//in this example, 14442. Integer Mp2 represents an equipment on mounting
//position 1 of the shelf; more can be added if required.
//Specify the root (base) equipment ID (rack) on which the final card is mounted.
container.setRootEquipmentId(14442);

//Specify an array of integers that represents mounting positions of the equipment
//hierarchy.
ArrayList mountingPositions = new ArrayList();
//Specify the shelf on mounting position of the base equipment 14442.
Integer Mp1 = new Integer(1);
//Specify an equipment on mounting position 1 of the shelf.
Integer Mp2 = new Integer(1);
mountingPositions.add(Mp1);
mountingPositions.add(Mp2);
container.setMountingPositions(mountingPositions);

//Specify the connection ECCKT or identifier to which you want to assign ports.
container.setCircuitIdentifier(" TEST/T1/AAAA1112/BBCYCAXF");

//Specify the port number of the equipment where the circuit must be assigned.
```

```
container.setPortSeq(6);
```

Creating Customer Connections (createCustomerConnection)

This API supports simultaneous creation of multiple customer connections based on the input data. If the custom attributes (CA) data is specified in the input, this API populates the CAs for connection. Otherwise, it just creates a connection.

You can also use the updateProvisioningInfo API for updating the provisioning information for the newly created circuit at the time of creating the circuit. To accomplish this, you must provide the input data relevant to the updateProvisioningInfo API. If provisioning data is passed in input, this API provisions the connection, which includes assignment of the created circuit to the specified parent connection and generating design lines based on the type of provisioning. Otherwise, it just creates a connection.

If the connection name is not provided in the input, it would be generated based on the input flag value (autoEcckt). If the flag is enabled, the ECCKT is generated. In this case, the ECCKT (connection name) generation is done based on the ECCKT type of the connection.

This API requires the customer account ID and product catalog ID for associating the newly created circuit to that customer account and tying it to the specified product catalog information.

This API supports:

- Regular circuit product.

This API does not support:

- Trunks and telephone number format circuit creation.
- Change/disconnect of labels and values.
- Template-based circuits.
- SONET/DLC assignments.
- Disconnected line blocks, which are managed manually in GUI through orders in MSS.
- SONET embedded in an optical network system.

EJB API Call

```
ConnectionAccessManagerRemote.createCustomerConnection(
   Connection[] connections,
   User user)
```

Container Object

```
com.metasolv.value.resource.entity.connection.Connection[]
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns an array list of successfully processed connections.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 2–10 shows the input parameters for the API.

Table 2–10 Input Parameters for the API

Input Parameter	Parameter Information
connections	**Data Type:** com.metasolv.value.resource.entity.connection.Connection[] **Description:** Contains information to run the functionality for creating connections. This array can contain input of connections. For example, to create ten connections as part of this API call, you must create an array with all the ten connection objects and pass it to the API. For example, suppose that for the third connection, the API creates the connection, adds labels and values, and populates custom attributes. However, if the input provisioning information is incorrect, the API fails and does not provision the connection. In this case, the following occur for the third connection: ■ The connection is created ■ The changes related to the creation of labels and values/custom attributes for the connection are committed ■ The changes related to the provisioning of the connection are rolled back ■ The API commits the changes related to the first, second, and third connections and continues to process the fourth connection, then the fifth connection, and so on If during the API processing, the creation of the third and sixth connections fails, the results object (which you get from the API return object) contains the errors related only to the third and sixth connections. The data related to all the remaining connections is imported/committed into the MSS database. This parameter is mandatory.
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.

Table 2–11 lists the containers you must set within the Connection container object.

Table 2–11 Containers To Set Within the Connection Container Object

Input Parameter	Data Type
productHierarchyValuesContainer	com.metasolv.value.resource.ProductHierarchyValuesContainer
labelValueContainer	com.metasolv.value.resource.LabelValueContainer
facilityConnection	com.metasolv.value.resource.entity.connection.FacilityConnection
provisioningInfoContainer	com.metasolv.value.resource.ProvisioningInfoContainer
caDataContainer	com.metasolv.value.resource.CaContainer

Note: If connection has to be provisioned after its creation, input data for provisioning should be provided in the Connection container itself.

Hence, the com.metasolv.value.resource.ProvisioningInfoContainer object should be set with relevant data for provisioning the circuit and this container should be provided in the input. The ProvisioningInfoContainer object is required in the input only when the connection must be provisioned with network assignment, facility assignment, and equipment assignment. If the ProvisioningInfoContainer object is not provided in the input, the connection is created, but it is not provisioned.

For non-facility connections, all input parameters should be set inside a container object of type com.metasolv.value.resource.entity.connection.Connection container. For facility connections, the container object is com.metasolv.value.resource.entity.connection.FacilityConnection. A facility connection contains few additional properties along with normal connection properties. The following tables include both the facility and non-facility connection properties.

Set the input parameters in the containers. Set the data inside the container object, and call the API by providing the container object in the input parameter.

Table 2–12 lists the input parameters that you set in the Connection object.

Table 2–12 Connection Container Input Parameters

Input Parameter	Parameter Information
eccktType	**Data Type:** com.metasolv.value.resource.entity.connection.EccktType **Description:** Specifies connection format. Valid values are: ■ **CLF**, **OTF** (usually for facility connections). ■ **CLS**, **OTS** (for specials). ■ Formats that start with CL are structured formats and remaining are free formats (OT*). This parameter is mandatory.
rateCode	**Data Type:** com.metasolv.value.resource.RateCode **Description:** Rate code of connection. This parameter is mandatory.
serviceType	**Data Type:** com.metasolv.value.service.ServiceType **Description:** You must set both the service type category and service type code in this object. This parameter is mandatory.
Connection name (eckt)	**Data Type:** String **Description:** The name of the connection. If the autoEcckt flag is false or off, this parameter is required; otherwise it is optional. This name should be unique.

Table 2–12 (Cont.) Connection Container Input Parameters

Input Parameter	Parameter Information
connectionType	**Data Type:** com.metasolv.value.resource.entity.connection.ConnectionType **Description:** Type of the connection. Valid values are: ■ **Facility** (F). ■ **Special** (S). This parameter is mandatory.
connectionStatus	**Data Type:** com.metasolv.value.resource.entity.connection.ConnectionStatus **Description:** Status of the connection. Valid value: Connection status = **6** (In Service) This parameter is mandatory.
ALocation	**Data Type:** com.metasolv.value.location.Location **Description:** Originating location ID/name of the connection. This parameter is mandatory.
ZLocation	**Data Type:** com.metasolv.value.location.Location **Description:** Terminating location ID/name. This parameter is mandatory.
jurisdictionCode	**Data Type:** com.metasolv.value.resource. JurisdictionCode **Description:** Specifies the jurisdiction code. This parameter is mandatory.
lineCoding	**Data Type:** String **Description:** Specifies the line coding algorithm for the circuit. This parameter is optional.
framing	**Data Type:** String **Description:** Specifies the framing algorithm for the circuit. This parameter is optional.
framingAnsi	**Data Type:** Char **Description:** Specifies the framing ANSI algorithm for the circuit. This parameter is optional.
customerCircuitDescription	**Data Type:** String **Description:** Specifies the customer circuit description. This parameter is optional.
networkChannelCode	**Data Type:** String **Description:** Specifies the network channel code. This value must exist in the MSS database. This parameter is optional.
networkChannelOptionCode	**Data Type:** String **Description:** Specifies the network channel option code. This must be a value that exists in MSS. This parameter is optional.

Table 2–12 (Cont.) Connection Container Input Parameters

Input Parameter	Parameter Information
partitionGroupName	**Data Type:** String **Description:** Name of the partition group. This parameter is mandatory if the partition group is selected in the **Active Partition Groups** preference.
customerAccountId	**Data Type:** Int **Description:** Identifies the customer to which the circuit created is associated. This parameter is mandatory.
productCatalogId	**Data Type:** Int **Description:** Identifies the product catalog of the circuit item to which the newly created circuit is associated. This parameter is mandatory.
designator	**Data Type:** String **Description:** This is required for creating a new facility connection with CLF format. It is used to generate the ECCKT. This parameter is optional.
telcoId	**Data Type:** String **Description:** This is required only when the connection format is CLS/CLT. It is used to generate ECCKT. This parameter is mandatory.
autoEcckt	**Data Type:** boolean **Description:** Specifies whether the connection ECCKT should be generated if it is not provided in input data. The ECCKT generation is done only when the autoEcckt flag is set to **true** and if the ECCKT is not provided in the input parameters. **True**: Generates the ECCKT **False** (default): Does not generate the ECCKT and logs an error if the ECCKT is not given in the input. This parameter is mandatory.

To add labels and values to the circuit at the circuit item level and parent product catalog level, the following parameters must be set in the com.metasolv.value.resource.ProductHierarchyValuesContainer object. Each ProductHierarchyValuesContainer object has a catalog ID and its corresponding labels and values array. If the parameters are not populated, the labels that are defined as mandatory in the product catalog for the parent and child are added and the default values are populated (if defined). These parameters are mandatory if the product catalog for the item and its parents has mandatory labels in it and does not have default values.

Table 2–13 lists the labels and values that you must set in the ProductHierarchyValuesContainer object. An array of these objects should be set in the connection object.

Table 2–13 ProductHierarchyValuesContainer Input Parameters

Input Parameter	Parameter Information
productCatalogId	**Data Type:** Int **Description:** Identifies the product catalog ID of the parent circuit product of the circuit item to which the newly created circuit is associated. This parameter is mandatory if the parent circuit product contains mandatory labels and values, and if the labels and values do not have default values. If this parameter is not provided for the mandatory scenario or if the catalog ID does not exist in MSS, the API fails and returns an error message.
labelValueContainer	**Data Type:** com.metasolv.value.resource.LabelValueContainer[] **Description:** Identifies the labels and values for the corresponding product catalog ID mentioned in the ProductHierarchyValuesContainer object. This parameter is mandatory if the parent circuit product contains mandatory labels and values, and if the labels and values do not have default values. If this parameter is not provided for the mandatory scenario or if the labels do not exist in MSS, the API fails and returns an error message. If the values are valid to the corresponding data type defined during label creation, then values are added; otherwise, the API fails and returns an error message.

To add the labels and values to the circuit, the parameters in Table 2–14 must be passed in the input. If these parameters are not populated, the labels that are defined as mandatory in the product catalog are added and the default values are populated (if defined). These parameters become mandatory if the product catalog has mandatory labels in it and does not have default values.

Table 2–14 lists the labels and values that you must set in the LabelValueContainer object. A vector containing these objects must be set in the connection object.

Table 2–14 LabelValueContainer Input Parameters

Input Parameter	Parameter Information
label	**Data Type:** String **Description:** Identifies the label to be added to the circuit item. This parameter is mandatory. If the label exists in the catalog, then it would be added; otherwise, the API fails and returns an error message.
value	**Data Type:** String **Description:** Identifies the value for the corresponding label. This parameter is mandatory. If the value is valid according to the corresponding data type defined during label creation, then values are added; otherwise, the API fails and returns an error message.
valueCode	**Data Type:** String **Description:** Specifies the value code. This parameter is mandatory. If the specified value code exists in the catalog, then it would be added; otherwise, the API fails and returns an error message.

Table 2–15 lists the parameters that you must set in the FacilityConnection object for facility connections.

Table 2–15 FacilityConnectionContainer Input Parameters

Input Parameter	Parameter Information
dedicatedToConnectionType	**Data Type:** String **Description:** Indicates whether this facility circuit is dedicated for special, trunk, or mixed circuits. Valid values are: ■ **M** (mixed) ■ **S** (special) ■ **T** (trunk) This parameter is optional, but is required for facility connections.
directTrunkQuantity	**Data Type:** Int **Description:** The number of access customer trunks riding this facility that are directly connected to either end office or tandem switches. This parameter is optional, but is required for facility connections. If this is not provided, it defaults to zero (0).
tdmDirectTrunkQuantity	**Data Type:** Int **Description:** The number of direct trunks on a transmission facility circuit that terminate on a tandem switch. This parameter is optional, but is required for facility connections. If this is not provided, it defaults to zero (0).

Table 2–16 lists the parameters that you must set in the CaContainer object to populate custom attributes (CAs) as part of circuit creation. This object represents only one CA. In case of multiple CAs, you must create an array of the CaContainer object. You must set the CaContainer object in the Connection object.

Table 2–16 CaContainer Object Input Parameters

Input Parameter	Parameter Information
conSpecificationName	**Data Type:** String **Description:** The name of the connection specification. This parameter is mandatory only if CaContainer is populated.
caName	**Data Type:** String **Description:** he name of the custom attribute (CA). This parameter is mandatory.
caValue	**Data Type:** String[] **Description:** Array list of CA values. If the CA type is multivalued, more than one CA value can be passed for a single CA. This parameter is mandatory.

Sample Data for Input Parameters

Example 2–5 is an example of the data values you must set for the input parameters to create a CLF type connection. The example also includes information to auto-generate the ECCKT. You set the setAutoEcckt() method to true to generate the connection identifier.

Example 2–5 Data Values for Input Parameters

```
//Specify the type of circuit identification you want to assign to the connection.
```

```
clsType = validValueAccessManager.getValue(
   com.metasolv.value.resource.entity.connection.EccktType.TYPE, "CLF");
//Specify the current operational status of the connection.
com.metasolv.value.resource.entity.connection.ConnectionStatus cs =
   validValueAccessManager.getValue(
   com.metasolv.value.resource.entity.connection.ConnectionStatus.TYPE,'6');

//Specify the originating location of the connection.
com.metasolv.value.location.Location Aloc =(
   com.metasolv.value.location.Location)valueFact.makeInstance(
      com.metasolv.value.location.Location.class);
Aloc.setLocationCode("CSVLILXE");

//Specify the terminating location of the connection.
com.metasolv.value.location.Location Zloc=(
   com.metasolv.value.location.Location)valueFact.makeInstance(
      com.metasolv.value.location.Location.class);
Zloc.setLocationId(91);

//Specify the service type for the connection. You must set both the service type
//category and service type code in this object.
com.metasolv.value.service.ServiceType st =(
   com.metasolv.value.service.ServiceType)valueFact.makeInstance(
      com.metasolv.value.service.ServiceType.class);
st.setServiceTypeCode("OC12");
st.setServiceTypeCategory("CLFI");

//Specify the rate code for the connection.
com.metasolv.value.resource.RateCode rc =
   (com.metasolv.value.resource.RateCode)valueFact.makeInstance(
      com.metasolv.value.resource.RateCode.class);
rc.setCode("OC12");

//Specify the type of the circuit you want to create.
com.metasolv.value.resource.entity.connection.ConnectionType ct =
   validValueAccessManager.getValue(
     com.metasolv.value.resource.entity.connection.ConnectionType.TYPE, 'F');

//Specify the jurisdiction code.
com.metasolv.value.resource.JurisdictionCode jc = (
   com.metasolv.value.resource.JurisdictionCode) validValueAccessManager.getValue(
     JurisdictionCode.TYPE,'1');

//Set the labels and values information for the connection.
LabelValueContainer lvc1 = new LabelValueContainer();
lvc1.setLabel("Label1");
lvc1.setValue("Value 1");
LabelValueContainer lvc2 = new LabelValueContainer();
lvc2.setLabel("AlphaLabel");
lvc2.setValue("Value alpha");
LabelValueContainer[] lvc = {lvc1,lvc2};
connection.setLabelValueContainer(lvc);

//The ProductCatalogId and CustomerAccountId parameters are mandatory for a
//customer connection.
connection.setProductCatalogId(13144);
connection.setCustomerAccountId(698207);

// Set the required information to auto-generate ECCKT.
// <prefix><service type><mod><serial number><suffix><Telco id><segment>
```

```
connection.setAutoEcckt(true);

//Set the required information to create the default connection positions.
connection.setCreateConnectionPositions(true);
```

Note: A new connection 103/OC12/CSVLILXE/NBRKILNT52T is created based on the specified data values.

Provisioning Assignments for Customer Circuits (updateProvisioningInfo)

The updateProvisioningInfo API enables you to provision:

- Network assignment (optical/SDH provisioning)
- Facility assignment
- Equipment assignment

updateProvisioningInfo

This API provisions the connection that includes assignment of the created circuit to the specified parent connection and generates design lines based on the type of provisioning.

This API performs the following operations:

- Add (add new blocks)
- Update (modify an existing block)
- Delete (delete an existing block)

Note: The existing equipment assignment API now supports virtual ports and EPAs.

EJB API Call

```
ConnectionAccessManagerRemote.updateProvisioningInfo(
   ProvisioningInfoContainer[] assignments,
   User user)
```

Container Object

```
com.metasolv.value.resource.ProvisioningInfoContainer[]
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns an array list of provisioned connections.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 2–17 shows the input parameters for the API.

Table 2–17 Input Parameters for the API

Input Parameter	Parameter Information
assignments	**Data Type:** com.metasolv.value.resource.ProvisioningInfoContainer[] **Description:** Contains information to run the functionality for provisioning connections. This array can contain input of connections. For example, to provision ten connections as part of this API call, you must create an array with all the ten connection objects and pass it to the API. For example, suppose that for the third connection, the API successfully completes network assignment and facility assignment. However, if the input provisioning information for equipment assignment is incorrect, the API does not complete the equipment assignment for this connection. In this case, the following occur for the third connection: ■ The changes related to the network assignment and facility assignment for the connection are committed ■ The changes related to the equipment assignment of the connection are rolled back ■ The API commits the changes related to the first, second, and third connections and continues to process the fourth connection, then the fifth connection, and so on This parameter is mandatory.
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.

Table 2–18 shows the containers you set in the ProvisioningInfoContainer object.

Table 2–18 Containers To Set Within the ProvisioningInfoContainer Object

Input Parameter	Data Type
networkAssignmentContainer	com.metasolv.value.resource.NetworkAssignmentContainer
channelAssignmentContainer	com.metasolv.value.resource.ChannelAssignmentContainer
portAssignmentContainer	com.metasolv.value.resource.PortAssignmentContainer

Note:

- NetworkAssignmentContainer can contain multiple ChannelAssignmentContainer and PortAssignmentContainer objects based on the provisioning requirement.
- ProvisioningInfoContainer can contain multiple NetworkAssignmentContainer objects (for optical network provisioning), ChannelAssignmentContainer objects (for facility assignments), and PortAssignmentContainer objects (for equipment assignments).

Note: Set the input parameters in the containers. Set the data inside the container object, and call the API by providing the container object in the input parameter.

Table 2–19 lists the input parameters that you set in the ProvisioningInfoContainer object.

Table 2–19 ProvisioningInfo Container Input Parameters

Input Parameter	Parameter Information
networkBlocks	**Data Type:** Vector **Description:** Collection of all the network assignment blocks. Size could be 0 or many. Contains the NetworkAssignmentContainer object. This parameter is optional. If this parameter is not provided, the API assumes that no optical provisioning must be done.
facilityBlocks	**Data Type:** Vector **Description:** Collection of all the facility assignment blocks. Size could be 0 or many. Contains the ChannelAssignmentContainer object. This parameter is optional. If this parameter is not provided, the API assumes that no facility assignment must be done.
equipmentBlocks	**Data Type:** Vector **Description:** List of all the PortAssignmentContainer objects. Could be 0 or many. This parameter is optional. If this parameter is not provided, API assumes that no equipment assignment must be done.
riderConnectionIdentifier	**Data Type:** String **Description:** Connection identifier of the child circuit that is getting provisioned. This parameter is mandatory when the updateProvisioningInfo API is called separately. If this parameter is not provided when the updateProvisioningInfo API is called separately, the API fails and returns an error message.
riderCircuitDesignId	**Data Type:** Long **Description:** Circuit design ID of child circuit/rider circuit that is getting provisioned. Will be retrieved based on connection identifier. Must be unique for the provided identifier. This parameter is optional.

Table 2–20 lists the input parameters that you set in the NetworkAssignmentContainer object.

Table 2–20 NetworkAssignment Container Input Parameters

Input Parameter	Parameter Information
channelAssignments	**Data Type:** Vector of ChannelAssignmentContainer objects **Description:** All channel assignment information; one element for one channel position. At least one element is mandatory. Network system should be set up properly according to the provided provisioning information; otherwise, the API fails to process.
portAssignments	**Data Type:** Vector of PortAssignmentContainer objects **Description:** All port assignment information, one element for one port assignment. This parameter is optional. Network system should be set up properly according to the provided provisioning information; otherwise, the API fails to process.
commonNetworkName	**Data Type:** String **Description:** Network system name to which the complete path belongs. This parameter is optional.
commonNetworkNsId	**Data Type:** Long **Description:** Network ID of common network system. This parameter is optional.
origCompName	**Data Type:** String **Description:** Originating component name. This parameter is mandatory. Must be provided if the ChannelAssignmentContainer object is used for network assignments. This parameter is not required for facility assignments.
origCompNbr	**Data Type:** Int **Description:** Originating component number. This parameter is optional. If this parameter is not provided, MSS assumes a value of zero.
termCompName	**Data Type:** String **Description:** Terminating component name. This parameter is mandatory. Must be provided if the ChannelAssignmentContainer object is used for network assignments, but is not required for facility assignments.

Table 2–20 (Cont.) NetworkAssignment Container Input Parameters

Input Parameter	Parameter Information
termCompNbr	**Data Type:** Int **Description:** Terminating component number. This parameter is optional. If this parameter is not provided, the API assumes a value of zero.
blockType	**Data Type:** String **Description:** This is the block type. Valid values are: **WP**: Working path **PP**: Protect path (implies WP + PP) **PPP**: Partial protect path This parameter is mandatory.
assignmentType	**Data Type:** String **Description:** This is the assignment type. Valid values are: **ADD** **UPDATE** **DELETE** This parameter is mandatory.

Table 2–21 lists the input parameters that you set in the ChannelAssignmentContainer object.

Table 2–21 ChannelAssignmentContainer Input Parameters

Input Parameter	Parameter Information
assignmentCode	**Data Type:** String **Description:** This is the assignment code. Valid values are: **A**: Assign. **U**: Unassign. (A null value is valid if read-only data is populated). This parameter is mandatory. For **Add** it is **A**. For **Remove/Delete** it is **U**. For **Update**, there are two ChannelAssignmentContainer objects: ▪ The first container object must be **U** (assignment that must be removed). ▪ The second container object must be **A** (new assignments that need to be made). This can be null while updating a network assignment when there are no changes in a particular segment. This cannot be null for facility assignments.
parentConnectionIdentifier	**Data Type:** String **Description:** Connection identifier of the parent. This parameter is mandatory. If this parameter is not provided when the ChannelAssignmentContainer object is populated, the API fails and returns an error message.
circuitPositions	**Data Type:** ArrayList **Description:** Circuit positions from the root parent circuit that is provided by Network Integrity. This parameter is mandatory.
protectPathIndicator	**Data Type:** String **Description:** This is the protect path indicator. Valid values are: **Y**: Yes (for protect path segment). **N**: No (for working path segment). **P**: For all partial protect path segments. This parameter is mandatory.

Table 2–22 lists the input parameters that you set in the PortAssignmentContainer object.

Table 2–22 PortAssignment Container Input Parameters

Input Parameter	Parameter Information
assign	**Data Type:** Boolean **Description:** To identify a port assign/unassign operation. Valid values are: **True**: Port assign. **False**: Unassign (this is the default). This parameter is mandatory. If this parameter is not provided, the API considers it as a port unassignment operation.
circuitIdentifier	**Data Type:** String **Description:** Exchange carrier circuit design ID (ECCKT) of the circuit in MSS. Useful to identify the connection. This parameter is mandatory.
rootEquipmentId	**Data Type:** Long **Description:** This represents base equipment (rack) on which the required CARD is mounted. Auto-generated ID in MSS. Unique identifier for equipment. This parameter is mandatory.
portSeq	**Data Type:** Long **Description:** Port on the equipment (CARD) to which the connection must be assigned. This parameter is mandatory.
mountingPositions	**Data Type:** ArrayList **Description:** List of mounting position numbers. This represents the position of the CARD related to the RACK (root equipment) to which the connection has to be assigned. This parameter is optional.
aZOtherCd	**Data Type:** String **Description:** Indicates the location where you must make the assignment. Valid values are: ■ **A** = A location ■ **Z** = Z location This parameter is optional.

Sample Data for Input Parameters

Example 2–6 shows an example of the data values you set for the input parameters to provision network assignments (optical/SDH provisioning), facility assignments, and equipment assignments.

Example 2–6 Data Values for Input Parameters

```
//Create an input instance for the channel assignment/unassignemnt container.
Vector<ChannelAssignmentContainer> channelcontainers = new
Vector<ChannelAssignmentContainer>();
ChannelAssignmentContainer channelAssignmentContainer1 = new
ChannelAssignmentContainer();
ChannelAssignmentContainer channelAssignmentContainer2 = new
ChannelAssignmentContainer();
ChannelAssignmentContainer channelAssignmentContainer3 = new
ChannelAssignmentContainer();
```

```
ChannelAssignmentContainer channelAssignmentContainer4 = new
ChannelAssignmentContainer();

//Provide information about the parent channels on which the rider connection
//should be assigned/unassigned.
channelAssignmentContainer1.setParentConnectionIdentifier("211  /VT1
/QLCNWAXARS0/RACNWI11ATM");
channelAssignmentContainer2.setParentConnectionIdentifier("211  /VT1
/RACNWI11ATM/TACMWAFA52T");
channelAssignmentContainer3.setParentConnectionIdentifier("211  /VT1
/QLCNWAXARS0/RACNWI11ATM");
channelAssignmentContainer4.setParentConnectionIdentifier("211  /VT1
/RACNWI11ATM/TACMWAFA52T");

//Provide information about the parent channel's channel position on which the
//rider connection will be assigned/unassigned.
ArrayList channelPos1 = new ArrayList();
channelPos1.add(new Integer(1));
ArrayList channelPos2 = new ArrayList();
channelPos2.add(new Integer(5));
ArrayList channelPos3 = new ArrayList();
channelPos3.add(new Integer(3));
ArrayList channelPos4 = new ArrayList();
channelPos4.add(new Integer(2));

//Specify whether to you want to assign or unassign the rider connection on the
//parent channel's channel position.
// N = working path,  Y = protected path
// U = unassign, " " = Assign
channelAssignmentContainer1.setCircuitPositions(channelPos1);
channelAssignmentContainer1.setProtectPathIndicator("N");
channelAssignmentContainer1.setAssignmentCode("U");
channelcontainers.add(channelAssignmentContainer1);
channelAssignmentContainer2.setCircuitPositions(channelPos2);
channelAssignmentContainer2.setProtectPathIndicator("N");
channelAssignmentContainer2.setAssignmentCode("U");
channelcontainers.add(channelAssignmentContainer2);
channelAssignmentContainer3.setCircuitPositions(channelPos3);
channelAssignmentContainer3.setProtectPathIndicator("N");
channelAssignmentContainer3.setAssignmentCode(" ");
channelcontainers.add(channelAssignmentContainer3);
channelAssignmentContainer4.setCircuitPositions(channelPos4);
channelAssignmentContainer4.setProtectPathIndicator("N");
channelAssignmentContainer4.setAssignmentCode(" ");
channelcontainers.add(channelAssignmentContainer4);

//Create the input instance for port assignment/unassignemnt container.
Vector<PortAssignmentContainer> portAssignmentContainers = new
Vector<PortAssignmentContainer>();
PortAssignmentContainer startContainerA = new PortAssignmentContainer(),
endContainerA = new PortAssignmentContainer();
PortAssignmentContainer startContainerU = new PortAssignmentContainer(),
endContainerU = new PortAssignmentContainer();

//Specify the location A (originating) and location Z (terminating) where you want
to unassign ports for the rider connection.
startContainerU.setPortSeq(1);
endContainerU.setPortSeq(4);

//Specify the location A (originating) and location Z (terminating) where you want
```

```
to assign ports for the rider connection.
startContainerA.setPortSeq(5);
endContainerA.setPortSeq(2);

//Specify the mounting position of the equipment on the root equipment where you
want to assign/unassign ports.
ArrayList<Integer> startMntPos = new ArrayList<Integer>();
startMntPos.add(1);
startMntPos.add(1);

//Specify the equipment's mounting position at location A (originating) on the
//root equipment where you want to unassign ports.
startContainerU.setMountingPositions(startMntPos);
startContainerU.setAZOtherCd("A");
startContainerU.setAssign(false);
startContainerU.setRootEquipmentId(636288);
startContainerU.setAssociatedToOrig(true);

//Specify the equipment's mounting position at location A (originating) on the
//root equipment where you want to assign ports.
startContainerA.setMountingPositions(startMntPos);
startContainerA.setAZOtherCd("A");
startContainerA.setAssign(true);
startContainerA.setRootEquipmentId(636288);
startContainerA.setAssociatedToOrig(true);
ArrayList<Integer> endMntPos = new ArrayList<Integer>();
endMntPos.add(1);
endMntPos.add(1);

//Specify the equipment's mounting position at location Z (terminating) on the
//root equipment where you want to unassign ports.
endContainerU.setMountingPositions(endMntPos);
endContainerU.setAZOtherCd("Z");
endContainerU.setAssign(false);
endContainerU.setRootEquipmentId(636297);
endContainerU.setAssociatedToOrig(false);

//Specify the equipment's mounting position at location Z (terminating) on the
//root equipment where you want to assign ports.
endContainerA.setMountingPositions(endMntPos);
endContainerA.setAZOtherCd("Z");
endContainerA.setAssign(true);
endContainerA.setRootEquipmentId(636297);
endContainerA.setAssociatedToOrig(false);

//Set the port assignment/un-assignment information in the port assignment vector.
portAssignmentContainers.add(startContainerU);
portAssignmentContainers.add(endContainerU);
portAssignmentContainers.add(startContainerA);
portAssignmentContainers.add(endContainerA);

//Create a vector of network assignment containers.
//ADD = add new blocks, UPDATE = modify an existing block, DELETE = delete an
//existing block
//WP = working path, PP = protect path, PPP = partial portect path
Vector<NetworkAssignmentContainer> nwContaienrs = new
Vector<NetworkAssignmentContainer>();
NetworkAssignmentContainer  nwContainer = new NetworkAssignmentContainer();
nwContainer.setAssignmentType("UPDATE");
nwContainer.setBlockType("WP");
```

```
nwContainer.setOrigCompName("Int_Comp7");
nwContainer.setTermCompName("Int_Comp10");
nwContainer.setPortAssignments(portAssignmentContainers);
nwContainer.setChannelAssignments(channelcontainers);
nwContaienrs.add(nwContainer);

//Create an instace of the provisioningInfoContainer.
ProvisioningInfoContainer provisioningInfoContainer = new
ProvisioningInfoContainer();

//Provide information about the rider connection.
provisioningInfoContainer.setRiderConnectionIdentifier("RAMS_DS1_7_CKT");

//Provide information about the network assignment container in the
provisioningInfoContainer.
provisioningInfoContainer.setNetworkBlocks(nwContaienrs);

//Set the provisioningInfoContainer in an array.
ProvisioningInfoContainer[] pic = new ProvisioningInfoContainer[1];
pic[0] = provisioningInfoContainer;

//Call the updateProvisioningInfo API on ConnectionAccessManagerRemote.
remote.updateProvisioningInfo(pic, user);

//Specify the following in situations where you want to do channel and equipment
assignments separately.
//provisioningInfoContainer.setFacilityBlocks();
//provisioningInfoContainer.setEquipmentBlocks();
```

Creating and Provisioning Customer Connections (createCustomerConnNew)

This API supports creation and provisioning of multiple customer connections at a time based on the input data.

You can also use the updateProvisioningInfo API for updating the provisioning information for the newly created circuit at the time of creating the circuit. To accomplish this, you must provide the input data relevant to the updateProvisioningInfo API. See "Provisioning Assignments for Customer Circuits (updateProvisioningInfo)" for more information.

If provisioning data is passed in input, this API provisions the connection, which includes assigning the created circuit to the specified parent connection and generating design lines based on the type of provisioning. If provisioning data is not passed in input, it just creates the connection. In cases where the API is expected to create and provision the connection and the provisioning fails, the connection is rolled back.

If the connection name is not provided in the input, it is generated based on the input flag value (autoEcckt). If the flag is enabled, the ECCKT is generated. The ECCKT (connection name) generation is based on the ECCKT type of the connection.

This API requires the customer account ID and product catalog ID for associating the newly created circuit to that customer account and tying it to the specified product catalog information.

This API supports:

- Regular circuit product.

This API does not support:

- Trunks and telephone number format circuit creation.
- Change/disconnect of labels and values.
- Template-based circuits.
- SONET/DLC assignments.
- Disconnected line blocks, which are managed manually in the UI through orders in MSS.
- SONET embedded in an optical network system.

EJB API Call

```
ConnectionAccessManagerRemote.createCustomerConnNew(
   Connection[] connections,
   User user)
```

Container Object

```
com.metasolv.value.resource.entity.connection.Connection[]
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns an array list of successfully processed connections.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 2–23 shows the input parameters for the API.

Table 2–23 Input Parameters for the API

Input Parameter	Parameter Information
connections	**Data Type:** com.metasolv.value.resource.entity.connection.Connection[] **Description:** Contains information to run the functionality for customer/non-customer virtual connections. This array can contain input for combination of connections. If you want to create ten virtual connections as part of this API call, you must create an array with all the ten connection objects and pass it to the API. For example, suppose that for the third connection, the API creates the connection, adds labels and values, and populates custom attributes. However, if the input provisioning information is incorrect, the API fails and does not provision the connection. In this case, all the changes are rolled back and the third connection is not created. The API commits the changes related to the first and second connections and continues to process the fourth connection, then the fifth connection, and so on. If during the API processing, the creation of the third and sixth connections fails, the results object (which you get from the API return object) contains the errors related only to the third and sixth connections. The data related to all the remaining connections is imported/committed into the MSS database. Each error message contains the index corresponding to the failed connection, including information about the reason for the failure. This parameter is mandatory.
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.

Table 2–24 shows the containers you set in the Connection container object.

Table 2–24 Containers To Set Within the Connection Container Object

Input Parameter	Data Type
productHierarchyValuesContainer	com.metasolv.value.resource.ProductHierarchyValuesContainer
labelValueContainer	com.metasolv.value.resource.LabelValueContainer
facilityConnection	com.metasolv.value.resource.entity.connection.FacilityConnection
provisioningInfoContainer	com.metasolv.value.resource.ProvisioningInfoContainer
caDataContainer	com.metasolv.value.resource.CaContainer

Table 2–25 lists the input parameters that you set in the Connection object.

Table 2–25 Connection Input Parameters

Input Parameter	Parameter Information
connectionCrossRef	**Data Type:** String **Description:** Identifies the cross-reference to be associated with the connection. This parameter is optional.
eccktType	**Data Type:** com.metasolv.value.resource.entity.connection.EccktType **Description:** Specifies the connection format. Valid values are: ■ **CLF**, **OTF** (usually for facility connections). ■ **CLS**, **OTS** (for specials). ■ Formats that start with CL are structured formats and remaining are free formats (OT*). This parameter is mandatory.
rateCode	**Data Type:** com.metasolv.value.resource.RateCode **Description:** Rate code of the connection. This parameter is mandatory.
serviceType	**Data Type:** com.metasolv.value.service.ServiceType **Description:** You must set both the service type category and service type code in this object. This parameter is mandatory.
connectionId	**Data Type:** String **Description:** ID to uniquely identify the connection. If the autoEcckt flag is turned off or false, this parameter is required; otherwise it is optional. This name should be unique.
connectionType	**Data Type:** com.metasolv.value.resource.entity.connection.ConnectionType **Description:** Type of the connection. Valid values are: ■ **F** (Facility) ■ **S** (Special) This parameter is mandatory.

Table 2–25 (Cont.) Connection Input Parameters

Input Parameter	Parameter Information
connectionStatus	**Data Type:** com.metasolv.value.resource.entity.connection.ConnectionStatus **Description:** Status of the connection. Valid value: ■ Connection status = **6** (In Service) This parameter is mandatory.
ALocation	**Data Type:** com.metasolv.value.location.Location **Description:** Originating location ID/name of the connection. This parameter is mandatory.
ZLocation	**Data Type:** com.metasolv.value.location.Location **Description:** Terminating location ID/name of the connection. This parameter is mandatory.
jurisdictionCode	**Data Type:** com.metasolv.value.resource. JurisdictionCode **Description:** Specifies the jurisdiction code. This parameter is mandatory.
lineCoding	**Data Type:** String **Description:** Specifies the line coding algorithm for the circuit. This parameter is optional.
framing	**Data Type:** String **Description:** Specifies the framing algorithm for the circuit. This parameter is optional.
framingAnsi	**Data Type:** Char **Description:** Specifies the framing ANSI algorithm for the circuit. This parameter is optional.
customerCircuitDescription	**Data Type:** String **Description:** Specifies the customer circuit description. This parameter is optional.
networkChannelCode	**Data Type:** String **Description:** Specifies the network channel code. This parameter is optional. If provided this value must be valid.
networkChannelOptionCode	**Data Type:** String **Description:** Specifies the network channel option code. This parameter is optional. If provided, this value must be valid.
partitionGroupName	**Data Type:** String **Description:** Name of the partition group. This parameter is mandatory if the partition group is selected in the **Active Partition Groups** preference.
customerAccountId	**Data Type:** Int **Description:** Identifies the customer to which the circuit created is associated. This parameter is mandatory.

Table 2–25 (Cont.) Connection Input Parameters

Input Parameter	Parameter Information
productCatalogId	**Data Type:** Int **Description:** Identifies the product catalog of the circuit item to which the newly created circuit is associated. This parameter is mandatory.
designator	**Data Type:** String **Description:** Used to auto-generate the ECCKT. This parameter is mandatory only in creating a new facility connection with CLF format.
telcoId	**Data Type:** String **Description:** Used to auto-generate the ECCKT. This parameter is mandatory only when the connection format is CLS.
eccktModifier	**Data Type:** String **Description:** Used to auto-generate the ECCKT. This parameter is mandatory only when the connection format is CLS/CLT.
autoEcckt	**Data Type:** boolean **Description:** Specifies whether the ECCKT should be auto-generated if it is not provided in the input data. **True**: Auto-generates the ECCKT. **False** (default): Does not auto-generate the ECCKT and logs an error if the ECCKT is not provided in the input. This parameter is mandatory.

To add labels and values to the circuit at the circuit item level and parent product catalog level, the parameters in Table 2–26 must be set in the com.metasolv.value.resource.ProductHierarchyValuesContainer object. Each ProductHierarchyValuesContainer object will have a catalog ID and its corresponding labels and values array. If these parameters are not populated, the labels that are defined as mandatory in the product catalog for the parent and child are added and the default values are populated (if defined). These parameters are mandatory if the product catalog for the item and its parents has mandatory labels in it and does not have default values.

Table 2–26 lists the labels and values that you must set in the ProductHierarchyValuesContainer object. An array of these objects should be set in the connection object.

Table 2–26 ProductHierarchyValuesContainer Input Parameters

Input Parameter	Parameter Information
productCatalogId	**Data Type:** String **Description:** Identifies the product catalog ID of the parent circuit product of the circuit item to which the newly created circuit is associated. This parameter is mandatory. If this parameter is not provided for the mandatory scenario or if the catalog ID does not exist in MSS, the API fails and returns an error message.
labelValueContainer	**Data Type:** com.metasolv.value.resource.LabelValueContainer[] **Description:** Identifies the labels and values for the corresponding product catalog ID mentioned in the ProductHierarchyValuesContainer object. This parameter is mandatory. If this parameter is not provided for the mandatory scenario or if the labels do not exist in MSS, the API fails and returns an error message. If the values are valid to the corresponding data type defined during label creation, then values are added; otherwise, the API fails and returns an error message.

To add the labels and values to the circuit, the parameters in Table 2–27 must be passed in the input. If these parameters are not populated, the labels that are defined as mandatory in the product catalog are added and the default values are populated (if defined). These parameters become mandatory if the product catalog has mandatory labels in it and does not have default values.

Table 2–27 lists the labels and values that you must set in LabelValueContainer object. A vector containing these objects must be set in the connection object.

Table 2–27 LabelValueContainer Input Parameters

Input Parameters	Parameter Information
label	**Data Type:** String **Description:** Identifies the label to be added to the circuit item. This parameter is mandatory. If the label exists in the catalog, then it is added; otherwise, the API fails and returns an error message.
value	**Data Type:** String **Description:** Identifies the value for the corresponding label. This parameter is mandatory. If the value is valid according to the corresponding data type defined during label creation, then values are added; otherwise, the API fails and returns an error message.
valueCode	**Data Type:** String **Description:** Specifies the value code. This parameter is mandatory. If the specified value code exists in the catalog, then it is added; otherwise, the API fails and returns an error message.

Table 2–28 lists the input parameters that you set in the FacilityConnection object for facility connections.

Table 2–28 FacilityConnection Input Parameters

Input Parameter	Parameter Information
dedicatedToConnectionType	**Data Type:** String **Description:** Indicates whether this facility circuit is dedicated for special, trunk, or mixed circuits. Valid values are: ■ **M** (Mixed) ■ **S** (Special) ■ **T** (Trunk) This parameter is optional, but is required for facility connections.
directTrunkQuantity	**Data Type:** Int **Description:** The number of access customer trunks riding this facility that are directly connected to either end office or tandem switches. This parameter is optional, but is required for facility connections. If this is not provided, it defaults to zero (0)
tdmDirectTrunkQuantity	**Data Type:** Int **Description:** The number of direct trunks on a transmission facility circuit that terminate on a tandem switch. This parameter is optional, but is required for facility connections. If this is not provided, it defaults to zero (0).

Sample Data for Input Parameters

Example 2–7 is an example of the data values you set for the input parameters to create a CLF type connection. The example also includes information to auto-generate the ECCKT. You set the setAutoEcckt() method to true to generate the connection identifier.

Example 2–7 Data Values for Input Parameters

```
//Specify the cross-reference to be associated with the connection.
connection.setConnectionCrossRef("Circuit123");

//Specify the type of circuit identification you want to assign to the connection.
clsType = validValueAccessManager.getValue(
   com.metasolv.value.resource.entity.connection.EccktType.TYPE, "CLF");

//Specify the current operational status of the connection.
com.metasolv.value.resource.entity.connection.ConnectionStatus cs =
   validValueAccessManager.getValue(
   com.metasolv.value.resource.entity.connection.ConnectionStatus.TYPE,'6');

//Specify the originating location of the connection.
com.metasolv.value.location.Location Aloc =(
   com.metasolv.value.location.Location)valueFact.makeInstance(
   com.metasolv.value.location.Location.class);
Aloc.setLocationCode("CSVLILXE");

//Specify the terminating location of the connection.
com.metasolv.value.location.Location Zloc=(
   com.metasolv.value.location.Location)valueFact.makeInstance(
   com.metasolv.value.location.Location.class);
Zloc.setLocationId(91);
```

```
//Specify the service type for the connection. You must set both the service type
//category and service type code in this object.
com.metasolv.value.service.ServiceType st =(
   com.metasolv.value.service.ServiceType)valueFact.makeInstance(
   com.metasolv.value.service.ServiceType.class);
st.setServiceTypeCode("OC12");
st.setServiceTypeCategory("CLFI");

//Specify the rate code for the connection.
com.metasolv.value.resource.RateCode rc =
   com.metasolv.value.resource.RateCode)valueFact.makeInstance(
   com.metasolv.value.resource.RateCode.class);
rc.setCode("OC12");

//Specify the type of the circuit you want to create.
com.metasolv.value.resource.entity.connection.ConnectionType ct =
   validValueAccessManager.getValue(
   com.metasolv.value.resource.entity.connection.ConnectionType.TYPE, 'F');

//Specify the jurisdiction code.
com.metasolv.value.resource.JurisdictionCode jc =  (
   com.metasolv.value.resource.JurisdictionCode)validValueAccessManager.getValue(
   JurisdictionCode.TYPE,'1');

//Set the labels and values information for the connection.
LabelValueContainer lvc1 = new LabelValueContainer();
lvc1.setLabel("Label1");
lvc1.setValue("Value 1");
LabelValueContainer lvc2 = new LabelValueContainer();
lvc2.setLabel("AlphaLabel");
lvc2.setValue("Value alpha");
LabelValueContainer[] lvc = {lvc1,lvc2};
connection.setLabelValueContainer(lvc);

//The ProductCatalogId and CustomerAccountId parameters are mandatory for a
//customer connection.
connection.setProductCatalogId(13144);
connection.setCustomerAccountId(698207);

//Set the required information to auto-generate ECCKT.
//<prefix><service type><mod><serial number><suffix><Telco id><segment>
connection.setAutoEcckt(true)

//Set the required information to create the default connection positions.
connection.setCreateConnectionPositions(true)
```

Note: A new connection 103/OC12/CSVLILXE/NBRKILNT52T is created based on the specified data values.

Creating Virtual Connections (createVirtualConnection)

The createVirtualConnection API supports enables you to create customer/non-customer virtual connections. If the required information is specified in the input, this API does the following:

- Creates a virtual connection
- Adds labels and values to the virtual connection

- Populates the custom attributes (CAs) for the virtual connection
- Provisions the virtual connection

If you provide the provisioning information for the virtual connection at the time of creating the connection, the createVirtualConnection does the following for the newly created virtual connection:

- Associates a connection specification with the new virtual connection
- Associates the new virtual connection with a network system
- Creates a new schematic design for the new virtual connection
- Allocates the custom attributes (CAs) for the new virtual connection for each segment in the schematic design

If provisioning data is not passed in the input, this API only creates the virtual connection. In cases where the API is expected to create and provision the connection and the provisioning fails, the connection creation process also fails and the virtual connection is rolled back.

If the connection name is not provided in the input, it would be generated based on the input flag value (autoEcckt). If the flag is enabled, the Exchange Carrier Circuit Identification (ECCKT) is generated. The ECCKT (circuit ID) generation is done based on the ECCKT type of the virtual connection.

This API requires the customer account ID and product catalog ID for associating the newly created circuit to that customer account and tying it to the specified product catalog information.

> **Note:** The customer account ID and product catalog ID are required only for customer connections.

This API supports only the following:

- Creation/provisioning of next-generation networks (NGN) virtual connections
- CONNECTOR product

This API does not support:

- Creation of emulated circuits

EJB API Call

```
ConnectionAccessManagerRemote.createVirtualConnection(
   Connection[] connections,
   User user)
```

Container Object

```
com.metasolv.value.resource.entity.connection.Connection[]
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns an array list of successfully creating connections.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 2–29 shows the input parameters for the API.

Table 2–29 Input Parameters for the API

Input Parameter	Parameter Information
connections	**Data Type:** com.metasolv.value.resource.entity.connection.Connection[] **Description:** Contains information to run the functionality for customer/non-customer virtual connections. This array can contain input for combination of customer/non-customer virtual connections. For virtual connections, you must set all input parameters in this Connection container. If the virtual connection has to be provisioned after its creation, input data for provisioning should be provided in the input. If you want to create ten virtual connections as part of this API call, you must create an array with all the ten connection objects and pass it to the API. For example, suppose that for the third connection, the API creates the connection, adds labels and values, and populates custom attributes. However, if the input provisioning information is incorrect, the API fails and does not provision the connection. In this case, all the changes are rolled back and the third connection is not created. The API commits the changes related to the first and second connections and continues to process the fourth connection, then the fifth connection, and so on. If during the API processing, the creation of the third and sixth connections fails, the results object (which you get from the API return object) contains the errors related only to the third and sixth connections. The data related to all the remaining connections is imported/committed into the MSS database. Each error message contains the index corresponding to the failed connection, including information about the reason for the failure. This parameter is mandatory.
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.

Table 2–30 lists the input parameters that you set in the Connection object.

Table 2–30 Connection Input Parameters

Input Parameter	Parameter Information
eccktType	**Data Type:** com.metasolv.value.resource.entity.connection.EccktType **Description:** Specifies the connection format. Valid values are: ■ **CLS**, **OTS** (for specials). ■ Structured formats start with CL and the remaining are free formats (OT*). This parameter is mandatory.
rateCode	**Data Type:** com.metasolv.value.resource.RateCode **Description:** Rate code of the connection. This parameter is optional.
serviceTypeCategory	**Data Type:** com.metasolv.value.service.ServiceType **Description:** Depends on the connection format. The serviceTypeCategory parameter is set within the serviceType object. Valid values are: ■ **CLCI-SS IntraLATA**, **CLCI-SS LATA Access** (for CLS format). ■ **CLCI-SS IntraLATA**, **CLCI-SS LATA Access**, **CLFI**, **CLCI-MSG**, **CLCI-SS Non Access** (OTS format). This parameter is mandatory.
serviceTypeCode	**Data Type:** com.metasolv.value.service.ServiceType **Description:** Specifies service provided by circuits, such as T3,T1, and so on. The serviceTypeCode parameter is set within the serviceType object. This parameter is mandatory.
name	**Data Type:** String **Description:** The name of the connection. This name should be unique. This parameter is mandatory if the autoEcckt flag is **false**.
conSpecificationName	**Data Type:** String **Description:** The name of the connection specification. This parameter is mandatory if the CaContainer is populated.
ALocation	**Data Type:** com.metasolv.value.location.Location **Description:** Originating location ID/location CLLI code (from the network_location table)/name of the connection. This parameter is mandatory.
ZLocation	**Data Type:** com.metasolv.value.location.Location **Description:** Terminating location ID/location CLLI code (from the network_location table)/name of the connection. This parameter is mandatory.
partitionGroupName	**Data Type:** String **Description:** Name of the partition group. This parameter is mandatory if the partition group is selected in the **Active Partition Groups** preference.

Table 2–30 (Cont.) Connection Input Parameters

Input Parameter	Parameter Information
jurisdictionCode	**Data Type:** com.metasolv.value.resource.JurisdictionCode **Description:** Specifies the jurisdiction code. This parameter is mandatory if the ECCKT type is of CLS format.
lineCoding	**Data Type:** String **Description:** Specifies the line coding algorithm for the circuit. This parameter is optional.
framing	**Data Type:** String **Description:** Specifies the framing algorithm for the circuit. This parameter is optional.
framingAnsi	**Data Type:** Char **Description:** Specifies the framing ANSI algorithm for the circuit. This parameter is optional.
customerCircuitDescription	**Data Type:** String **Description:** Specifies the customer circuit description. This parameter is optional.
customerAccountId	**Data Type:** int **Description:** Identifies the customer to which the circuit created is associated. This parameter is mandatory for customer connections.
productCatalogId	**Data Type:** int **Description:** Identifies the product catalog of the circuit item to which the newly created circuit is associated. This parameter is mandatory for customer connections if Customer Account Id is provided.

Table 2–30 (Cont.) Connection Input Parameters

Input Parameter	Parameter Information
autoEcckt	**Data Type:** boolean **Description:** Specifies whether connection ECCKT should be auto-generated if it is not provided in input data. Valid values are: **True**: Generates ECCKT **False** (the default): Does not generate ECCKT and logs an error if the name parameter is not given in the input. This parameter is optional. When creating a new connection, if you do not provide the connection name (ECCKT) in the input, you must set the autoEcckt flag to **true**, which auto-generates the ECCKT based on the connection format: CLS or OTS. The telcoId and eccktModifier parameters are mandatory if the autoEcckt flag is set to **true**. You must set the following system preferences to auto-generate ECCKT: ■ **Constant for Freeformat Circuit ID:** In the **Constant for Freeformat Circuit ID** field, specify a value for the constant. This preference automatically generates the ECCKT with a combination of the constant and the next available number according to existing connection IDs. ■ **Auto-Generate Freeformat Serial Number:** Determines whether serial numbers for CLS-format/OTS-format circuits are generated sequentially. When this preference is set to **Y**, one of the following occurs depending on the circuit type: – For CLS-format (Structured) circuits, the ECCKT is automatically generated with the next available number according to existing connection IDs. – For OTS-format (Freeformat-Serial) circuits, the ECCKT is automatically generated with a combination of the constant for free-format connection ID and the next available number according to existing connection IDs. The **Constant for Freeformat Circuit ID** and **Auto-Generate Freeformat Serial Number** preferences are located under Preferences - Service Request - Connection in the MetaSolv Solution application
telcoId	**Data Type:** String **Description:** Used to auto-generate the ECCKT. This parameter is mandatory if the connection format is CLS and if the autoEcckt flag is set to **true**. When this preference is set to Y, one of the following happens depending on the circuit format:
eccktModifier	**Data Type:** String **Description:** Used to auto-generate the ECCKT. This parameter is mandatory if the connection format is CLS and if the autoEcckt flag is set to **true**.

Table 2–30 (Cont.) Connection Input Parameters

Input Parameter	Parameter Information
labelValueContainer	**Data Type:** Array of LabelValueContainer objects. **Description:** Contains information of labels and values for the connection. You must set this container within the Connection container. If any of the labels and values are defined as required on the product catalog item and not specified in the input, the API fails and returns an error. This parameter is mandatory if the connection is a customer connection and the product catalog item has required labels and values. See Table 2–31, " LabelValueContainer Input Parameters" for more information.
caDataContainer	**Data Type:** Array of CaContainer objects. **Description:** Contains information to populate custom attributes (CAs) for the connection. You must set this container within the Connection container. If the connection specification is not associated to the connection and if you try to populates CAs, the API fails and returns an error. This parameter is mandatory. See Table 2–32, " CaContainer Input Parameters" for more information.
virtualProvisioningInfoContainer	**Data Type:** VirtualProvisioningInfoContainer **Description:** Contains information to provision the new virtual connection. The VirtualProvisioningInfoContainer object is required in the input only if you are provisioning the virtual connection. If the VirtualProvisioningInfoContainer object is not provided within the Connection container, the virtual connection is created, but it is not provisioned. This parameter is optional. See Table 2–33, " VirtualProvisioningInfoContainer Input Parameters" for more information.

If you want to add the labels and values to the virtual connection, the parameters in Table 2–31 must be passed in the input. If these parameters are not populated, the labels that are defined as mandatory in the product catalog are added and the default values are populated (if defined). These parameters become mandatory if the product catalog has mandatory labels in it and does not have default values.

Note: The **LabelValueContainer** object is valid only if the Product Catalog Id is populated.

Table 2–31 lists the labels and values that you must set in the LabelValueContainer object. A vector containing these objects must be set within the Connection object.

Table 2–31 LabelValueContainer Input Parameters

Input Parameter	Description
label	**Data Type:** String **Description:** Identifies the label to be added to the circuit item. This parameter is mandatory. If the label exists in the catalog, then it would be added; otherwise, the API fails and returns an error message.
value	**Data Type:** String **Description:** Identifies the value for the corresponding label. This parameter is mandatory. If the value is valid according to the corresponding data type defined during label creation, then values are added; otherwise, the API fails and returns an error message.
valueCode	**Data Type:** String **Description:** Specifies the value code. This parameter is mandatory. If the specified value code exists in the catalog, then it would be added; otherwise, the API fails and returns an error message.

Table 2–32 lists the parameters that you must set in the CaContainer object to populate custom attributes (CA) as part of circuit creation. This object represents only one CA. In case of multiple CAs, you must create an array of the CaContainer object. You must set the CaContainer object within the Connection object.

Table 2–32 CaContainer Input Parameters

Input Parameter	Description
caName	**Data Type:** String **Description:** The name of the custom attribute (CA). This parameter is mandatory.
caValue	**Data Type:** String[] **Description:** Array list of CA values. If the CA type is multivalued, more than one CA value can be passed for a single CA. This parameter is mandatory.

Table 2–33 lists the input parameters that you set in the VirtualProvisioningInfoContainer object.

Table 2–33 VirtualProvisioningInfoContainer Input Parameters

Input Parameter	Description
connectionEcckt	**Data Type:** String **Description:** Connection ECCKT or identifier to uniquely differentiate the connection. This parameter is mandatory.
networkSystemName	**Data Type:** String **Description:** Represents the network system name where the embedded network systems exist. Useful when the parent and child components exist in embedded network systems and those embedded network systems are shared by multiple network systems. This parameter is mandatory.
connectionSpecName	**Data Type:** String **Description:** Connection specification name with which the connection must be associated. This parameter is mandatory if custom attribute (CA) values are not populated on the connection using createVirtualConnection API.
parentCompName	**Data Type:** String **Description:** Component name at A location. This parameter is mandatory.
parentCompNum	**Data Type:** Int **Description:** Number specified on the parent component. Used to differentiate between multiple components that have the same name. This parameter is optional.
parentCompNsName	**Data Type:** String **Description:** Represents the network system name of the parent component. It is required when the network is embedded in the other network system. This parameter is mandatory if the parent component is shared by multiple network systems.
childCompName	**Data Type:** String **Description:** Component name at Z location. The given connection would be associated between component A and component Z. This parameter is mandatory.
childCompNum	**Data Type:** Int **Description:** Number specified on the child component. Used to differentiate between multiple components that have the same name. This parameter is optional.
childCompNsName	**Data Type:** String **Description:** Represents the network system name of the child component. It is required if the child network is embedded in the other network system. This parameter is mandatory if the child component is shared by multiple network systems.

If you want to create the schematic design for the new virtual connection, provide the input information about the physical connections on which the virtual connection should ride.

Populate an array of SchematicDesignContainer objects with the parameter listed in Table 2–34 and set it in the VirtualProvisioningInfoContainer object to populate the schematic design information.

Table 2–34 SchematicDesignContainer Input Parameter Description

Input Parameter	Description
parentEcckt	**Data Type:** String **Description:** ECCKT of the physical connections on which the virtual connection should ride. This parameter is mandatory.

Table 2–35 lists the parameters you must set in the AllocationCaContainer object to populate allocation custom attributes (CAs) at a segment level in the schematic design. To populate the allocation CAs at all the segment levels of the schematic design, create an array of the AllocationCaContainer object in the input.

The information about allocation CAs is required to associate them at each segment level in the schematic design. This input information is contained in the AllocationCaContainer object and it must be set within the VirtualProvisioningInfoContainer object.

The prerequisite to populate allocation CAs is that the virtual connection should contain schematic design information.

Table 2–35 AllocationCaContainer Input Parameters

Input Parameter	Description
circuitEcckt	**Data Type:** String **Description:** Specifies the physical connection ECCKT on which the virtual circuit should ride. This parameter is mandatory.
caContainer	**Data Type:** CaContainer object **Description:** Array list of CaContainer objects. This parameter is mandatory if AllocationCaContainer is populated. See Table 2–32, " CaContainer Input Parameters" for more information.

Sample Data for Input Parameters

Example 2–8 and Example 2–9 are examples of the data values you must set for the input parameters to create and provision a customer connection of CLS format. The example also includes information to auto-generate the ECCKT.

Example 2–8 Data Values for Input Parameters to Create a Virtual Connection

```
// Create the connection input instance.
com.metasolv.value.resource.entity.connection.Connection conn = (
   com.metasolv.value.resource.entity.connection.Connection)valueFact.
      makeInstance(
      com.metasolv.value.resource.entity.connection.Connection.class);

// Set the following parameters in the connection input instance:

// Set the ECCKT type.
com.metasolv.value.resource.entity.connection.EccktType ett =
   validValueAccessManager.getValue(
```

```
   com.metasolv.value.resource.entity.connection.Ecc  ktType.TYPE, "CLS");
conn.setEccktType(ett);

// Create the service type instance and set the service type code and service type
// category in the service type instance.
com.metasolv.value.service.ServiceType st = (
   com.metasolv.value.service.ServiceType)valueFact.makeInstance(
   com.metasolv.value.service.ServiceType.class);
st.setServiceTypeCode("HC");
st.setServiceTypeCategory("CLCI-SS LATA Access");
conn.setServiceType(st);

// Create location instances and set the location ID.
com.metasolv.value.location.Location A = (
      com.metasolv.value.location.Location)valueFact.makeInstance(
      com.metasolv.value.location.Location.class);
// Specify either the location ID or the location code.
A.setLocationId(1497648); // Location ID where the circuit originates.
A.setLocationCode("CUSTOMER LOC");
conn.setALocation(A);
com.metasolv.value.location.Location Z = (
   com.metasolv.value.location.Location)valueFact.makeInstance(
      com.metasolv.value.location.Location.class);
// Specify either the location ID or the location code.
Z.setLocationId(1325689); // Location ID where the circuit terminates.
Z.setLocationCode("A1PRAVL1");
conn.setZLocation(Z);

// Create the rate code instance.
com.metasolv.value.resource.RateCode rc = (
   com.metasolv.value.resource.RateCode)valueFact.makeInstance(
     com.metasolv.value.resource.RateCode.class);
rc.setCode("DS0");
conn.setRateCode(rc);

// Create the jurisdiction code instance and set it in the connection input
// instance.
com.metasolv.value.resource.JurisdictionCode jc = (
   com.metasolv.value.resource.JurisdictionCode)validValueAccessManager.getValue(
   JurisdictionCode.TYPE, '1');
conn.setJurisdictionCode(jc);

// Set the required information to auto-generate ECCKT.
// CLS ECCKT type format:
// <prefix><service type><mod><serial number><suffix><Telco id><segment>
// If autoEcckt flag is set to true, the TelcoId and EccktModifier parameters are
// mandatory.
conn.setAutoEcckt(true);
conn.setTelcoId("AAAA");
conn.setEccktModifier("VC");

// Set the product catalog ID and customer parameters in the connection input
// instance.
// The ProductCatalogId and CustomerAccountId parameters are mandatory for a
// customer connection.
conn.setProductCatalogId(18574);
conn.setCustomerAccountId(1922524);

// The following parameters are optional for a customer connection.
conn.setCustomerCircuitDescription("Oracle DSL Circuit");
```

```
conn.setLineCoding("DWMT"); // Defined on the product item.
conn.setFraming("ESF"); // Defined on the product item.
conn.setFramingAnsi("Y");  // If not provided, the default value is N.

// Provide the partition group name.
// The partition group name must exist in the Active Pratition Groups preference.
// The Use Partition Level Security for Access to inventory Data preference must
// be set to Y.

// Set the labels and values information for the virtual connection.
LabelValueContainer lvca = new LabelValueContainer();
LabelValueContainer[] lvcb = new LabelValueContainer[1];
lvca.setLabel("Test Label");
lvca.setValue("My Label");
lvcb[0] = lvca;
conn.setLabelValueContainer(lvcb);

// Set the custom attributes (CAs) information for the connection:

// Create the CaContainer instances.
CaContainer caContainer = new CaContainer();
CaContainer caContainer1 = new CaContainer();
CaContainer caContainer2 = new CaContainer();
String[] val1 = new String[1];
String[] val2 = new String[1];
String[] val3 = new String[2];

// Set the information related to a specific CA in the CaContainer.

caContainer.setCaName("Virtual Broadband Service Category");
val1[0] = "MPLS";
caContainer.setCaValue(val1);
caContainer1.setCaName("Data Speed");
val2[0] = "10";
caContainer1.setCaValue(val2);
caContainer1.setCaUom("M");
caContainer2.setCaName("Related Connection Id");
val3[0] = "Connection Id 1";
val3[1] = "Connection Id 2";
caContainer2.setCaValue(val3);

// Set the CaContainer instances in the CaContainer array.
CaContainer caList[] = { caContainer, caContainer1, caContainer2 };

// Set the custom attributes (CAs) information for the connection.
conn.setCaDataContainer(caList);

// Set the connection specification for the virtual connection.
// The following parameter is mandatory only if the custom attributes information
// is provided for the virtual connection.
conn.setConSpecificationName("Internet Connection");
```

Example 2–9 Data Values for Input Parameters to Provision the Virtual Connection

```
// Set the provisioning information for the connection.
VirtualProvisioningInfoContainer vpiContainer = new
VirtualProvisioningInfoContainer();

// Associate the virtual connection to a network system.
// Specify the originating component for the virtual connection.
```

```
vpiContainer.setParentCompName("DSL CS12");
vpiContainer.setParentCompNsName("NMK DSL NS");
// Specify the terminating component for the virtual connection.
vpiContainer.setChildCompName("NMK DSLAM");
vpiContainer.setChildCompNsName("NMK Em DSL NS");
// The following parameters are optional in this scenario:
// vpiContainer.setNetworkSystemName("NMK DSL NS");
// vpiContainer.setParentCompNum(1);
// vpiContainer.setChildCompNum(2);

// Set the schematic design information for the virtual connection.
// In the SchematicDesignContainer object, provide the information about the
// connection on which the virtual connection should ride.
// If the virtual connection rides on multiple connections, set multiple
// SchematicDesignContainer objects for each connection.
ArrayList<SchematicDesignContainer> schematicDesignContainers = new
ArrayList<SchematicDesignContainer>();
SchematicDesignContainer schematicDesignContainer1 = new
SchematicDesignContainer();
schematicDesignContainer1.setParentEcckt("PHY2 ABL IP");
schematicDesignContainers.add(schematicDesignContainer1);
SchematicDesignContainer schematicDesignContainer2 = new
SchematicDesignContainer();
schematicDesignContainer2.setParentEcckt("PHYCN/10G   /GAFBMTXC   /AAPKOKXA   ");
schematicDesignContainers.add(schematicDesignContainer2);

// Set the SchematicDesignContainer object in the VirtualProvisioningInfoContainer
// object.
vpiContainer.setSchematicDesignContainer(schematicDesignContainers);

// Set the allocation custom attributes (CAs) information.
// Create the AllocationCaContainer instance to set the information about CAs.
AllocationCaContainer allocationCaContainer = new AllocationCaContainer();
allocationCaContainer.setCircuitEcckt("PHYCN/10G   /GAFBMTXC   /AAPKOKXA   ");

// Set the information about each CA in the CaContainer.
CaContainer caContainer3 = new CaContainer();
caContainer3.setCaName("VPI");
String[] caVal4 =  {"151"};
caContainer3.setCaValue(caVal4);
CaContainer caContainer4 = new CaContainer();
caContainer4.setCaName("VCI");
String[] caVal5 =  {"510"};
caContainer4.setCaValue(caVal5);

// Set all the CaContainers in an array.
ArrayList<CaContainer> allocationCaValuesContainer = new ArrayList<CaContainer>();
allocationCaValuesContainer.add(caContainer3);
allocationCaValuesContainer.add(caContainer4);

// Set the CaContainers array in the AllocationCaContainer object.
allocationCaContainer.setCaContainer(allocationCaValuesContainer);

// Set the AllocationCaContainer object in the AllocationCaContainers array.
ArrayList<AllocationCaContainer> allocationCaValuesContainers = new
ArrayList<AllocationCaContainer>();
allocationCaValuesContainers.add(allocationCaContainer);

// Set the AllocationCaContainer object in the VirtualProvisioningInfoContainer
// object.
```

```
vpiContainer.setAllocationCaContainer(allocationCaValuesContainers);

// Set the VirtualProvisioningInfoContainer object in the Connection container.
conn.setVirtualProvisioningInfoContainer(vpiContainer);
```

Logging Messages

This section describes some of the logging messages that are logged during an API call, which is helpful when trying to determine the success or failure of an API call.

Table 2–36 lists an example of the situation where the API logs a success message based on your actions.

Table 2–36 API Success Message

Action	Message
Providing the required information in the input.	`For Connection at index: 0, connection created with Identifier HM/HCVC/73175 /  /AAAA/.`

Table 2–37 examples of situations where the API logs failure message based on your actions.

Table 2–37 API Failure Messages

Action	Message
Providing incorrect connection input data.	`Error code: 260426, Error message : For connection at index 0, Given telcoId DEF ,for connection at index 0 does not exists in database. Error code: 260478, Error message : For connection at index 0, Invalid Line Coding ABC. Error code: 260479, Error message : For connection at index 0, Invalid Framing XYZ.`
Not providing the mandatory labels and values in the input.	`Error code: 260465, Error message : For connection at index 0, Label and Value is required for the catalog id 18574, for the connection HM/HCVC/73177 /  /AAAA/.`
Associating an invalid CA value to a connection.	`Error code: 260451, Error message : For connection at index 0, Custom Attribute Broadband Service Category is not associated to given connection spec Internet Connection in Utilities. Please configure it accordingly.`
Providing invalid data when associating a virtual connection to a network system.	`Error code: 64329, Error message : For connection at index 0, Parent component name DSL CS12 is invalid, for connection identifier HM/HCVC/73181 / /AAAA/ . Error code: 64333, Error message : For connection at index 0, Network system name SRK Em DSL NS12 for child component is invalid, for connection identifier HM/HCVC/73181 / /AAAA/.`
Providing an invalid connection as part of the schematic design.	`Error code: 75199, Error message : For connection at index 0, PHY2 ABL IP 123 is a invalid connection and cannot be used to create the schematic design`
Providing a duplicate value to an allocation CA value that is defined to not accept duplicate values for a connection.	`Error code: 75116, Error message : For connection at index 0, This assignment is a duplicate.  The following connection exists with the selected value. Virtual Connection: HM/HCVC/73175 /  /AAAA/  , Parent Connection: PHYCN/10G  /GAFBMTXC  /AAPKOKXA.`

Provisioning Virtual Connections (provisionVirtualConnection)

The provisionVirtualConnection API enables you to update the provisioning information for non-channelized (virtual) connections. If the required information is specified in the input, this API does the following:

- Associates a connection specification to the new virtual connection
- Associates the new virtual connection to a network system
- Creates or updates the schematic design for the existing virtual connection
- Populates or updates the allocation custom attributes (CAs) for the existing virtual connection for each segment in the schematic design

If the mandatory data for provisioning is not passed in the input, the API fails and does not update the virtual connection.

For example, consider a scenario where the API is expected to associate the existing virtual connection to a network system and also create/update the schematic design of the virtual connection. In this scenario, if the process of creating/updating the schematic design fails, the process of associating the connection to the network system also fails and all the modifications to the virtual connection are rolled back.

This API supports only the following:

- Creation/provisioning of next-generation networks (NGN) virtual connections (CONNECTOR product)

This API does not support the following:

- Creation/update of emulated circuits

EJB API Call

```
ConnectionAccessManagerRemote.provisionVirtualConnection(
   VirtualProvisioningInfoContainer[] vpics,
   User user)
```

Container Object

```
com.metasolv.value.resource.VirtualProvisioningInfoContainer[]
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns an array list of successfully processed connections.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 2–38 shows the input parameters for the API.

Table 2–38 Input Parameters for the API

Input Parameter	Parameter Information
vpics	**Data Type:** com.metasolv.value.resource.VirtualProvisioningInfoContainer[] **Description:** Contains information to update the functionality of customer/non-customer existing virtual connections. This array can contain input for combination of customer/non-customer virtual connections. If you want to update ten virtual connections as part of this API call, you must create an array with all the ten provisioning objects and pass it to the API. For example, suppose that for the third connection, the API updates the connection, associated to a network system, and created/updated the schematic design. However, if the input provisioning information to update the allocation parameters information is incorrect, the API fails and does not provision the connection. In this case, all the changes are rolled back and the third connection is not updated. The API commits the changes related to the first and second connections and continues to process the fourth connection, then the fifth connection, and so on. If during the API processing, the updating of the third and sixth connections fails, the results object (which you get from the API return object) contains the errors related only to the third and sixth connections. The data related to all the remaining connections is imported/committed into the MSS database. Each error message contains the index corresponding to the failed connection, including information about the reason for the failure. This parameter is mandatory.
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.

Table 2–39 shows the containers you set with the VirtualProvisioingInfoContainer.

Table 2–39 Containers To Set Within the VirtualProvisioningInfoContainer Object

Input Parameter	Data Type
schematicDesignContainer	com.metasolv.value.resource.SchematicDesignContainer
allocationCaContainer	com.metasolv.value.resource.AllocationCaContainer

Maintaining Design Information for Virtual Connections

This section provides information about how and when the provisionVirtualConnection API creates new issues for virtual connections.

Scenarios When New Issues are Created

The provisionVirtualConnection API creates issues to maintain a history of the design changes made to the virtual connection.

The API creates new issues in the following scenarios:

- After the virtual connection is created, you pass the input provisioning information only to populate allocation CAs. In this case, the API creates a new issue and copies the updated information to the newly created issue.
- You pass the input provisioning information to update the existing schematic design for the virtual connection. In this case, the API creates a new issue and copies the updated information to the newly created issue.

- You pass the input provisioning information to update the existing allocation custom attributes (CA) for the virtual connection. In this case, the API creates a new issue and copies the updated information to the newly created issue.

  See "How New Issues are Created" for more information.

The API does not create new issues in the following scenarios:

- After the virtual connection is created, you pass all the input provisioning information, including information to associate the connection with a network system, create a new schematic design, and populate allocation CAs. In this case, the API does not create a new issue and copies all the updated information to the **Current** issue.
- After the virtual connection is created, you pass the input provisioning information only to associate the connection with the a network system. In this case, the API does not create a new issue and copies the new data to the **Current** issue. After you associate the connection with the network system, you cannot update the network system information.
- After the virtual connection is created, you pass the input provisioning information only to create a schematic design. In this case, the API does not create a new issue and copies the new data to the **Current** issue.

How New Issues are Created

The following scenarios explain how the provisionVirtualConnection API creates issues for the virtual connection you are updating.

Scenario 1

You are updating either a non-ordered connection or an ordered connection with all tasks completed.

Table 2–40 lists the existing issues for the connection before you make an API call.

Table 2–40 Issues For the Connection Before the API Call

Issue	Issue Status	Order Number
1	Current	1416623

Table 2–41 lists the new issues for the connection after you make an API call.

Table 2–41 Issues For the Connection After the API Call

Issue	Issue Status	Order Number
2	Current	1416623
1	Previous	1416623

In this scenario, the following occurs:

- The status of the existing **Current** (#1) issue changes to **Previous** (#1)
- New **Current** (#2) issue is created
- The updated connection information is copied to the highest issue (#2)

Scenario 2

You are updating an ordered connection with the Design Layout Report Date (DLRD) task not completed.

Table 2–42 lists the existing issues for the connection before you make an API call.

Table 2–42 Issues For the Connection Before the API Call

Issue	Issue Status	Order Number
1	Pending	1416624

Table 2–43 lists the new issues for the connection after you make an API call.

Table 2–43 Issues For the Connection After the API Call

Issue	Issue Status	Order Number
2	Pending	1416624
1	Overridden	1416624

In this scenario, the following occurs:

- The status of the existing **Pending** (#1) issue changes to **Overridden** (#1)
- New **Pending** (#2) issue is created
- The updated connection information is copied to the highest issue (#2)

Scenario 3

You are updating an ordered connection as part of a change order with all tasks completed.

Table 2–44 lists the existing issues for the connection before you make an API call.

Table 2–44 Issues For the Connection Before the API Call

Issue	Issue Status	Order Number
2	Pending	1416624
1	Current	1416623

Table 2–45 lists the new issues for the connection after you make an API call.

Table 2–45 Issues For the Connection After the API Call

Issue	Issue Status	Order Number
4	Pending	1416624
3	Current	1416623
2	Overridden	1416624
1	Previous	1416623

In this scenario, the following occurs:

- From Table 2–44 the statuses of both the existing **Current** (#1) and **Pending** (#2) issues change to **Previous** (#1) and **Overridden** (#2) respectively, shown in Table 2–45.
- New **Current** (#3) and **Pending** (#4) issues are created, shown in Table 2–45.
- The updated connection information is copied to the highest issues (#3 and #4).

Note: For every newly created issue, the API creates design notes that describe why a particular issue was created. For example, Issue 2 is created via NI/MSS resolution.

Input Parameters to Be Set in Containers

The following tables include information about the input parameters that you must set in containers for virtual connection provisioning.

If you want to provision the virtual connection after it is created, you must provide input data for provisioning in the VirtualProvisioningInfoContainer object.

Table 2–46 lists the input parameters that you set in the VirtualProvisioningInfoContainer object.

Table 2–46 VirtualProvisioningInfoContainer Input Parameters

Input Parameter	Description
connectionEcckt	**Data Type:** String **Description:** Connection ECCKT or identifier to uniquely differentiate the connection. This parameter is mandatory.
networkSystemName	**Data Type:** String **Description:** Represents the network system name where the embedded network systems exist. This parameter is mandatory only when the parent and child components exist in embedded network systems and those embedded network systems are shared by multiple network systems.
connectionSpecName	**Data Type:** String **Description:** Connection specification name with which the connection must be associated. This parameter is mandatory if custom attribute (CA) values are not populated on the connection using createVirtualConnection API and want to associate the virtual connection to a network system.
parentCompName	**Data Type:** String **Description:** Component name at A location. This parameter is mandatory.
parentCompNum	**Data Type:** int **Description:** Number specified on the parent component. Used to differentiate between multiple components that have the same name. This parameter is optional.
parentCompNsName	**Data Type:** String **Description:** Represents the network system name of the parent component. It is required when the network is embedded in the other network system. This parameter is mandatory if the parent component is shared by multiple network systems.
childCompName	**Data Type:** String **Description:** Component name at Z location. The given connection would be associated between component A and component Z. This parameter is mandatory.

Table 2–46 (Cont.) VirtualProvisioningInfoContainer Input Parameters

Input Parameter	Description
childCompNum	**Data Type:** int **Description:** Number specified on the child component. Used to differentiate between multiple components that have the same name. This parameter is optional.
childCompNsName	**Data Type:** String **Description:** Represents the network system name of the child component. This parameter is mandatory if the child component is shared by multiple network systems.
schematicDesignContainer	**Data Type:** Array of SchematicDesignContainer objects **Description:** This parameter is mandatory if you want to create or update the schematic design for the existing virtual connection. See Table 2–47, " SchematicDesignContainer Input Parameter Description" for more information.
allocationCaContainer	**Data Type:** Array of AllocationCaContainer objects **Description:** This parameter is mandatory if you want to populate or update the allocation custom attributes (CAs) for the existing virtual connection at each segment in the schematic design. See Table 2–48, " AllocationCaContainer Input Parameters" for more information.

If you want to create or update the schematic design for the existing virtual connection, provide the input information about the physical connections on which the virtual connection should ride.

Populate an array of SchematicDesignContainer objects with the parameter listed in Table 2–47 and set it in the VirtualProvisioningInfoContainer object to populate the schematic design information.

Table 2–47 SchematicDesignContainer Input Parameter Description

Input Parameter	Description
parentEcckt	**Data Type:** String **Description:** Used to either associate the connection with the schematic design or to unassociate the connection from the schematic design depending on whether or not the virtual circuit is already associated with the schematic design. If the virtual connection has a schematic design, the connection value provided in this parameter is unassociated or replaced with replacement parent ECCKT's in the schematic design. See "replacementParentEcckts" for more information. If the virtual connection does not have a schematic design, the connection value that you provide in this parameter is associated to the virtual connection. This parameter is mandatory.
replacementParentEcckts	**Data Type:** Array of String objects **Description:** The list of connections provided in this array list replaces the ECCKT value specified for the parent ECCKT in the SchematicDesignContainer. See "parentEcckt" for more information. This parameter is mandatory depending on the scenario it is used.

Note: If the input information in the API contains multiple SchematicDesignContainer containers, you must specify replacementParentEcckts for either all the containers or for none of them.

Table 2–48 lists the parameters that you must set in the AllocationCaContainer object to populate allocation custom attributes (CAs) at a segment level in the schematic design. To populate the allocation CAs at all the segment levels of the schematic design, create an array of the AllocationCaContainer object in the input.

The information about allocation CAs is required based on your configuration to associate them at each segment level in the schematic design. The allocation CAs information contained in the AllocationCaContainer object must be set within the VirtualProvisioningInfoContainer object.

The prerequisites to populate allocation CAs are as follows:

- The virtual connection should contain schematic design
- Allocation CAs should already be configured

Table 2–48 AllocationCaContainer Input Parameters

Input Parameter	Description
circuitEcckt	**Data Type:** String **Description:** Specifies the physical connection ECCKT on which the virtual circuit should ride. This parameter is mandatory.
caContainer	**Data Type:** Array of CaContainer objects **Description:** This parameter is mandatory if AllocationCaContainer is populated. See Table 2–49 for more information.

Table 2–49 lists the parameters that you must set in the CaContainer object to populate custom attributes (CA) as part of circuit creation. This object represents only one CA. In case of multiple CAs, you must create an array of the CaContainer objects. You must set the CaContainer objects within the AllocationCaContainer object.

Table 2–49 CaContainer Input Parameters

Input Parameter	Description
caName	**Data Type:** String **Description:** The name of the custom attribute (CA). This parameter is mandatory.
caValue	**Data Type:** String[] **Description:** Array list of CA values. If the CA type is multivalued, more than one CA value can be passed for a single CA. This parameter is mandatory.

Sample Data for Input Parameters

Example 2–10 is an example of the data values that you must set in the input parameters to provision the virtual connection.

Example 2–10 Data Values for Input Parameters

```
//Create the input array for VirtualProvisioningInfoContainer incstances.
VirtualProvisioningInfoContainer[] pic = VirtualProvisioningInfoContainer[1];

//Create the input VirtualProvisioningInfoContainer instance.
VirtualProvisioningInfoContainer vpic = VirtualProvisioningInfoContainer();

//Set the following parameters in the VirtualProvisioningInfoContainer input
//instance to associate the virtual connection to a network system:
//Set the Connection Identifier type.
vpic.setConnectionEcckt("HM/HCVC/70912 /   /AAAA/   ");

//Set connection specification name.
vpic.setConnectionSpecName("Internet Connection");

//Set the parent network system only if the same parent component is shared
//between multiple network systems.
//vpic.setParentCompNsName("DSK DSL NS");

//Set the parent component number only if multiple components have the same name
//in the same network system.
//vpic.setParentCompNum(2);

//Set the parent component name.
vpic.setParentCompName("DSL CS");

//Set the child component name.
vpic.setChildCompName("DSK DSLAM");

//Set the child component number only if multiple components have the same name in
//the same network system.
//vpic.setChildCompNum(2);

//Set the child network system only if the same child component is shared between
//multiple network systems.
//vpic.setChildCompNsName("DSK Em DSL NS");

//Set the schematic design information for the virtual connection.
//In the SchematicDesignContainer object, provide the information about the
//connection on which the virtual connection should ride.
//If the virtual connection rides on multiple connections, set multiple
//SchematicDesignContainer objects for each connection.
ArrayList<SchematicDesignContainer> schematicDesignContainers = new
ArrayList<SchematicDesignContainer>();
SchematicDesignContainer schematicDesignContainer1 = new
SchematicDesignContainer();
schematicDesignContainer1.setParentEcckt("CPHY3/T1    /DDCTMN01   /AAPKOKXA   ");

//The replacement ECCKT information is required only when updating the existing
//schematic design information.
//In this example, " CPHY3/T1    /DDCTMN01   /AAPKOKXA   " connection will be
//replaced with "  /HC--/000001/   /AAAA/   " connection in the schematic design.
//This information is not required when creating the schematic design.
ArrayList<String> replacementEcckts1 = new ArrayList<String>();
replacementEcckts1.add("  /HC--/000001/   /AAAA/   ");
schematicDesignContainer1.setReplacementParentEcckts(replacementEcckts1);

SchematicDesignContainer schematicDesignContainer2 = new
SchematicDesignContainer();
```

```
schematicDesignContainer2.setParentEcckt("PHYCN/10G   /GAFBMTXC   /AAPKOKXA   ");

//The replacement ECCKT information is required only when updating the existing
//schematic design information.
//In this example, " PHYCN/10G   /GAFBMTXC   /AAPKOKXA   " connection will be
//replaced with "  /HC--/000002/   /AAAA/   " connection in the schematic design.
//This information is not required when creating the schematic design.
ArrayList<String> replacementEcckts2 = new ArrayList<String>();
replacementEcckts2.add("  /HC--/000002/   /AAAA/   ");
schematicDesignContainer2. setReplacementParentEcckts(replacementEcckts2);

//Set the schematicDesignContainers into the schematicDesignContainer array.
schematicDesignContainers.add(schematicDesignContainer1);
schematicDesignContainers.add(schematicDesignContainer2);

//Set the SchematicDesignContainer object in the VirtualProvisioningInfoContainer
//object.
vpic.setSchematicDesignContainer(schematicDesignContainers);

//Create or update the allocation custom attributes (CAs) information.
//Create the AllocationCaContainer instance to set the information about CAs.
AllocationCaContainer allocationCaContainer = new AllocationCaContainer();

//Case 1: If you want to create the schematic design, set the value of any of the
//parentEcckt to which you want to set the allocation parameters.
//Case 2: If you want to update the existing schematic design, set the value of
//any replacementEcckt to which you want to set the allocation parameters.
//Case 3: If you want to retain the existing schematic design and change only the
//allocation parameters of a connection, set the value of the connection for which
//you want to modify the allocation parameters.
allocationCaContainer.setCircuitEcckt("PHYCN/10G   /GAFBMTXC   /AAPKOKXA   ");

//Set the information about each CA in the CaContainer.
CaContainer caContainer3 = new CaContainer();
caContainer3.setCaName("VPI");
String[] caVal4 =  {"151"};
caContainer3.setCaValue(caVal4);
CaContainer caContainer4 = new CaContainer();
caContainer4.setCaName("VCI");
String[] caVal5 =  {"510"};
caContainer4.setCaValue(caVal5);

//Set all the CaContainers in an array.
ArrayList<CaContainer> allocationCaValuesContainer = new ArrayList<CaContainer>();
allocationCaValuesContainer.add(caContainer3);
allocationCaValuesContainer.add(caContainer4);

//Set the CaContainers array in the AllocationCaContainer object.
allocationCaContainer.setCaContainer(allocationCaValuesContainer);

//Set the AllocationCaContainer object in the AllocationCaContainers array.
ArrayList<AllocationCaContainer> allocationCaValuesContainers = new
ArrayList<AllocationCaContainer>();
allocationCaValuesContainers.add(allocationCaContainer);

//Set the AllocationCaContainer object in the VirtualProvisioningInfoContainer
//object.
vpic.setAllocationCaContainer(allocationCaValuesContainers);
```

Updating Connections (updateConnection)

This API supports updating multiple connections. Each connection is updated based on the input data provided for it. This API supports updating the following connection types:

- Special
- Virtual
- Facility
- Bandwidth

This API does not update auto-build connections. This API updates only the non-ordered connections.

Using this API, you can update the following data for a connection:

- Rate code
- Location A
- Location Z
- Number of channel positions
- Channel information
- Admin information
- General properties
- User data
- CAs

The **connectionIdentifier** parameter is the only parameter that is mandatory. This API updates the information as part of the connection's latest issue. This API does not create a new issue.

EJB API Call

```
ConnectionAccessManagerRemote.updateConnection(
   UpdateConnectionContainer[] containers,
   User user)
```

Container Object

```
com.metasolv.value.resource.UpdateConnectionContainer[]
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns an array list of successfully updated connections.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 2–50 shows the input parameters for the API.

Table 2–50 Input Parameters for the API

Input Parameter	Parameter Information
containers	**Data Type:** com.metasolv.value.resource.UpdateConnectionContainer[] **Description:** Contains information to run the functionality to update connections. This array can contain input for connections. If you want to update ten connections as part of this API call, you must create an array with all the ten connection objects and pass it to the API. For example, if the third connection is not updated due to any incorrect input data, the API commits the data related to the first and second connections and rollbacks the changes for only the third connection and continues to process the fourth connection, then the fifth connection, and so on. This parameter is mandatory.
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.

Table 2–51 lists the input parameters that you set in the UpdateConnectionContainer object.

Table 2–51 UpdateConnectionContainer Input Parameters

Input Parameter	Parameter Information
connectionIdentifier	**Data Type:** String **Description:** Connection ECCKT or identifier to uniquely identify the connection. This parameter is mandatory.
rateCode	**Data Type:** String **Description:** Rate code of the connection. This parameter is optional.
locationA	**Data Type:** String **Description:** Originating location of the connection. This parameter is optional.
locationZ	**Data Type:** String **Description:** Terminating location of the connection. This parameter is optional.
totalNumberOfChannelPositions	**Data Type:** String **Description:** The number of channel positions for a facility or special connection. Use this value to increase or decrease the number of channel positions for a connection. This parameter is optional.
channelPositionsRateCode	**Data Type:** String **Description:** The channel positions that the rate code needs while increasing the channel positions from zero to a positive number. This parameter is optional.

Table 2–51 (Cont.) UpdateConnectionContainer Input Parameters

Input Parameter	Parameter Information
channelInfo	**Data Type:** Java.util.ArrayList **Description:** Contains ChannelContainer objects. Each ChannelContainer object contains channel-level information to be updated. This parameter is optional.
circuitLayoutOrder	**Data Type:** String **Description:** Circuit layout order for the connection. This parameter is optional.
remarksLine1	**Data Type:** String **Description:** Remarks line1 for the connection. This parameter is optional.
remarksLine2	**Data Type:** String **Description:** Remarks line2 for the connection. This parameter is optional.
remarksLine3	**Data Type:** String **Description:** Remarks line3 for the connection. This parameter is optional.
ecDesignContact	**Data Type:** String **Description:** Ec design contact for the connection. This parameter is optional.
ecTelephoneNumber	**Data Type:** String **Description:** Ec telephone number for the connection. This parameter is optional.
ecMaintainControlOffice	**Data Type:** String **Description:** Ec maintain control office value for the connection. This parameter is optional.
ecOperationsControlOffice	**Data Type:** String **Description:** Ec operations control office value for the connection. This parameter is optional.
designNotes	**Data Type:** String **Description:** Design notes for a connection. This parameter is optional.
designContactNbr	**Data Type:** String **Description:** Design contact number for a connection. This parameter is optional.
designContactTelNbr	**Data Type:** String **Description:** Design contact telephone number for a connection. This parameter is optional.
framing	**Data Type:** String **Description:** Framing value for the connection. This parameter is optional.

Table 2–51 (Cont.) UpdateConnectionContainer Input Parameters

Input Parameter	Parameter Information
framingANSI	**Data Type:** String **Description:** Framing ANSI value for the connection. This parameter is optional.
lineCoding	**Data Type:** String **Description:** Line coding value for the connection. This parameter is optional.
dedicatedTo	**Data Type:** String **Description:** The "Dedicated To" value for the connection. This parameter is optional.
officialCompanyUse	**Data Type:** Char **Description:** Official company use value for the connection. This parameter is optional.
protectedConnection	**Data Type:** Char **Description:** Protected connection value for the connection. This parameter is optional.
hostRemoteIndicator	**Data Type:** Char **Description:** Host remote indicator value for the connection. This parameter is optional.
allowLowerRates	**Data Type:** Char **Description:** Allow lower rates value for the connection. This parameter is optional.
redesignCandidateIndicator	**Data Type:** Char **Description:** Redesign candidate indicator value for the connection. This parameter is optional.
redesignReasonCode	**Data Type:** String **Description:** Redesign reason code value for the connection. This parameter is optional.
userData	**Data Type:** Java.util.HashMap **Description:** Contains name and value pairs. Name represents the name of the user field and value represents the value for the field. This parameter is optional.
caValues	**Data Type:** Java.util.ArrayList **Description:** Contains list of CaUpdateContainers. Each ca update container contains the information related to a CA to be updated. This parameter is optional.

Sample Data for Input Parameters

Example 2–11 is an example of the data values you must set for the input parameters to update a facility connection type.

Example 2–11 Data Values for Input Parameters

```
//Create the update connection input instance.
UpdateConnectionContainer ucc = new UpdateConnectionContainer();

//Specify the channel information to be updated for facility/special connections.
ucc .connectionIdentifier = "TEST /OC12 /AAAA1112   /BBCYCAXF";

//Specify the rate code for the connection.
ucc .rateCode = "OC12";

//Specify the originating location of the connection.
ucc .locationA = "AAAA1234";

//Specify the terminating location of the connection.
ucc .locationZ = "BBBB1234";

//Specify the channel information to be updated for facility/special connections.
ucc .totalNumberOfChannelPositions = "2";
ucc .channelPositionsRateCode = "DS1";
//(This is required only if channel positions are 0 and you want to increase them)
ArrayList channelInfo = new ArrayList();
ChannelContainer cc = new ChannelContainer();
cc.parentChannelIdentifier = "TEST /OC12 /AAAA1112   /BBCYCAXF";
cc.channelPosition = "2";
cc.channelDesignation = "CT2";
cc.channelRatecode = "DS1";
cc.channelRemarks = "Test Remarks";
channelInfo.add(cc);
ucc .channelInfo(channelInfo);

//Specify the information to update connection information under Connection Design
//Summary - Additional Details - Admin tab.
ucc. circuitLayoutOrder = "Order 4";
ucc. remarksLine1 = "Line1";
ucc. remarksLine2 = "Line2";
ucc. remarksLine3 = "Line3";
ucc. ecDesignContact = "James";
ucc. ecTelephoneNumber = "12345";
ucc. ecMaintainControlOffice = "67890";
ucc. ecOperationsControlOffice = "45678";
ucc.designNotes = "Test Notes";
ucc. designContactNbr = "12345";
ucc. designContactTelNbr = "67890";

//Specify the information to update connection information under Connection Design
//Summary - Properties - General tab.
ucc. Framing = "M13";
ucc. framingANSI = "Y";
ucc. lineCoding = "AMI";
ucc. dedicatedTo = "Special";
ucc. officialCompanyUse = "Y";
ucc.protectedConnection = "Y";
ucc.hostRemoteIndicator = "Y";
ucc.allowLowerRates = "Y";
ucc.redesignCandidateIndicator = "Y";
ucc.redesignReasonCode = "Element Added to a network";

//Specify the information to update connection information under Connection Design
//Summary - Properties - User Data tab.
HashMap userData = new HashMap();
userData.put("Originating Lata", "test");
```

```
userData.put("Diversity-1","last");
ucc. setUserData(userData);

//Specify the information to update connection information under Connection Design
//Summary - Properties - Custom Attributes tab.
ArrayList cas = new ArrayList();
CaUpdateContainer cuc = new CaUpdateContainer();
cuc.caName = "Bit Rate";
cuc.caUom = "M";
ArrayList valuesList = new ArrayList();
CaValueContainer caValueContainer = new CaValueContainer();
caValueContainer.caValue = "1.544";
valuesList.add(caValueContainer);
cuc. setCaValue(valuesList);
cas.add(cuc);
ucc. setCaValues(cas);
```

3

Working with Engineering Work Orders

This chapter provides information about the Engineering Work Order (EWO) APIs.

Creating a Work Order (createWorkOrder)

This API supports creation of an Engineering Work Order based on the input data provided.

EJB API Call

```
WorkOrderManagerRemote.createWorkOrder (User user, WorkOrder workOrder)
```

Container Object

```
com.metasolv.value.order.WorkOrder
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns the successfully created work order.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 3–1 shows the input parameters for the API.

Table 3–1 Input Parameters for the API

Input Parameter	Parameter Information
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.
workOrder	**Data Type:** com.metasolv.value.order.WorkOrder **Description:** Contains information to create an Engineering Work Order. This parameter is mandatory.

Many of these Java objects are not instantiated directly. Instead, you create these objects using the factory objects like GenericFactory. Sample code creating objects using the factory methods is shown in Example 3–1.

Table 3–2 lists the input parameters that you set in the WorkOrder container object. Each parameter in the table indicates whether it is optional or mandatory in the container object for this API.

Table 3–2 WorkOrder Input Parameters

Input Parameter	Parameter Information
documentNumber	**Data Type:** int **Description:** Document number of the order. For new orders the business logic generates this parameter. This value should be zero for a create. This parameter is optional.
description	**Data Type:** String **Description:** Description provided for the order. This parameter is optional.
dueDate	**Data Type:** java.util.Calendar **Description:** Due date of the order. If this is provided then it cannot be in the past. This parameter is optional. If this parameters is not provided this field will be set to the current date.
projectKey	**Data Type:** com.metasolv.value.EntityKey **Description:** Project ID of the order. This parameter is optional.
referenceId	**Data Type:** String **Description:** Reference ID of the order. This parameter is optional.
responsiblePerson	**Data Type:** String **Description:** Responsible person name of the order. This parameter is optional.
status	**Data Type:** com.metasolv.value.WorkOrderStatus **Description:** Status of the order. This parameter is mandatory.
orderNumber	**Data Type:** String **Description:** The order number. If this parameter is populated, it must be unique, for example, no other EWOs exist with the same work order number. Also, the length must not exceed 17 characters. If this parameter is not input, it is automatically generated by the application. This parameter is optional.
isAutoWorkOrder **Note**: This parameter is obsolete.	**Data Type:** boolean **Description:** This parameter is obsolete and must be set to **false**. This parameter is optional.

Sample Data for Input Parameters

Example 3–1 is an example of the data values you must set for the input parameters to create a Engineering Work Order.

Example 3–1 Data Values for Input Parameters

```
protected static ValueFactory valueFact;
private ValidValueAccessManager validValueAccessManager;
```

```
// create the value factory
valueFact = GenericFactory.makeValueFactory();

// create the valid value access manager
validValueAccessManager = GenericFactory.makeValidValueAccessManager(userContext);

// create the work order object using the value factory.
WorkOrder wo = (WorkOrder)valueFact.makeInstance(WorkOrder.class);

// specify the order number
wo.setOrderNumber("RS EWO API12");

// specify the due date
wo.setDueDate(Calendar.getInstance());

// specify the description
wo.setDescription("RS API Created EWO12");

// specify the responsible person
wo.setResponsiblePerson("RSMITH");

// specify the reference id
wo.setReferenceId("Ref Id");

// create the status object
WorkOrderStatus woStatus =
validValueAccessManager.getValue(WorkOrderStatus.TYPE,WorkOrderStatus.IN_
PROGRESS);

// specify the status
wo.setStatus(woStatus);

// creating the project key object using the value factory and
// setting the values in it.
ProjectKey proKey = (ProjectKey) valueFact.makeInstance(ProjectKey.class);
proKey.setDocumentNumber(638178);
```

Updating a Work Order (updateWorkOrder)

This API supports updating of an existing Engineering Work Order based on the input data provided.

EJB API Call

```
WorkOrderManagerRemote.updateWorkOrder (User user, WorkOrder workOrder)
```

Container Object

```
com.metasolv.value.order.WorkOrder
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns the successfully updated work order.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 3–3 shows the input parameters for the API.

Table 3–3 Input Parameters for the API

Input Parameter	Parameter Information
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.
workOrder	**Data Type:** com.metasolv.value.order.WorkOrder **Description:** Contains information to create an Engineering Work Order. This parameter is mandatory.

Table 3–4 lists the input parameters that you set in the WorkOrder container object. Each parameter in the table indicates whether it is optional or mandatory in the container object.

Table 3–4 Input Parameters

Input Parameter	Parameter Information
documentNumber	**Data Type:** int **Description:** Document number of the order. This parameter must be provided when updating an existing order. This parameter is mandatory.
description	**Data Type:** String **Description:** Description provided for the order. This parameter is optional.
dueDate	**Data Type:** java.util.Calendar **Description:** Due date of the order. If this is provided then it cannot be in the past. This parameter is optional. If this parameter is not provided, this field will be set to the current date.
supplementType	**Data Type:** com.metasolv.value.order. WorkOrderSupplementType **Description:** The type of the order supplement. This parameter is optional.
projectKey **Note**: This parameter is obsolete.	**Data Type:** com.metasolv.value.EntityKey **Description:** Project ID of the order. This parameter is optional.
referenceId	**Data Type:** String **Description:** Reference ID of the order. This parameter is optional.
responsiblePerson	**Data Type:** String **Description:** Responsible person name of the order. This parameter is optional.

Table 3–4 (Cont.) Input Parameters

Input Parameter	Parameter Information
status	**Data Type:** com.metasolv.value.WorkOrderStatus **Description:** Status of the order. This parameter is optional.
orderNumber	**Data Type:** String **Description:** The order number. If this parameter is populated, it must be unique, for example, no other EWOs exist with the same work order number. Also, the length must not exceed 17 characters. This parameter is optional.
isAutoWorkOrder **Note**: This parameter is obsolete.	**Data Type:** boolean **Description:** This parameter is obsolete and must be set to **false**. This parameter is optional.

Sample Data for Input Parameters

Example 3–2 is an example of the data values you must set for the input parameters to update a Engineering Work Order.

Example 3–2 Data Values for Input Parameters

```
protected static ValueFactory valueFact;
private ValidValueAccessManager validValueAccessManager;

// create the value factory
valueFact = GenericFactory.makeValueFactory();

// create the valid value access manager
validValueAccessManager = GenericFactory.makeValidValueAccessManager(userContext);

// Create the work order object using the value factory.
WorkOrder wo = (WorkOrder) valueFact.makeInstance(WorkOrder.class);

// Specify the document number to which we need to update the order
wo.setDocumentNumber(408979);

// Specify the order number
wo.setOrderNumber("RS EWO API12");

// Specify the due date
wo.setDueDate(Calendar.getInstance());

// Specify the description
wo.setDescription("RS API Created EWO12");

// Specify the responsible person
wo.setResponsiblePerson("RSMITH");

// Specify the reference id
wo.setReferenceId("Ref Id");

// Create the supplement type
WorkOrderSupplementType wost =
validValueAccessManager.getValue(WorkOrderSupplementType.TYPE, "2");

// Specify the supplement type
```

```
wo.setSupplementType(wost);
```

Creating a Work Order Note (createWorkOrderNote)

This API supports the creation of an Engineering Work Order note based on the input data provided.

EJB API Call

```
WorkOrderManagerRemote.createWorkOrderNote (User user, OrderNote orderNote)
```

Container Object

```
com.metasolv.value.order.OrderNote
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns the successfully created work order note.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 3–5 shows the input parameters for the API.

Table 3–5 Input Parameters for the API

Input Parameter	Parameter Information
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.
orderNote	**Data Type:** com.metasolv.value.order.OrderNote **Description:** Contains information to create an Engineering Work Order Note. This parameter is mandatory.

Table 3–6 lists the object you need to set within the OrderNote container object.

Table 3–6 Containers To Set Within the OrderNote Container Object

Input Parameter	Data Type
orderKey	com.metasolv.value.EntityKey

Table 3–7 lists the input parameters you set in the OrderNote container object. Each parameter in the table indicates whether it is optional or mandatory in the container object for this API.

Table 3–7 Input Parameters for the API

Input Parameter	Parameter Information
note	**Data Type:** String **Description:** Note information. This parameter is mandatory.
dateEntered	**Data Type:** java.util.Calendar **Description:** Date of the note entered. This parameter is optional. If this parameter is not provided, this field will be set to the current date.
orderKey	**Data Type:** com.metasolv.value.EntityKey **Description:** Order key for the note. This parameter is mandatory.
noteNumber	**Data Type:** String **Description:** A numerical string reference of the note. You use this when more than one note is associated with the same entity. This parameter is optional.
systemGeneratedInd	**Data Type:** String **Description:** System indicator of the note. This is for internal use only. This parameter is internal.
userId	**Data Type:** String **Description:** Userid who created the note. This parameter is optional.

Sample Data for Input Parameters

Example 3–3 is an example of the data you must set for the input parameters to create an Engineering Work Order note.

Example 3–3 Data Values for Input Parameters

```
protected static ValueFactory valueFact;
private ValidValueAccessManager validValueAccessManager;

// create the value factory
valueFact = GenericFactory.makeValueFactory();

// create the valid value access manager
validValueAccessManager =
   GenericFactory.makeValidValueAccessManager(userContext);

// create the order key object using the value factory and setting the
// information on it.
OrderKey orderKey = (OrderKey)valueFactory.makeInstance(OrderKey.class);
orderKey.setDocumentNumber(9533773);

// create the order note object using the value factory and setting the
// information on it.
OrderNote orderNote = (OrderNote) valueFact.makeInstance(OrderNote.class);
orderNote.setNote("Test EWO Notes");
orderNote.setNoteNumber("1");
orderNote.setOrderKey(orderKey);
orderNote.setUserId("ASAP");
```

Updating a Work Order Notes (updateWorkOrderNote)

This API supports updating an Engineering Work Order note based on the input data provided.

EJB API Call

```
WorkOrderManagerRemote.updateWorkOrderNote (User user, OrderNote orderNote)
```

Container Object

```
com.metasolv.value.order.OrderNote
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns the successfully updated work order note.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 3–8 shows the input parameters for the API.

Table 3–8 Input Parameters for the API

Input Parameter	Parameter Information
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.
orderNote	**Data Type:** com.metasolv.value.order.OrderNote **Description:** Contains information to update an Engineering Work Order Note. This parameter is mandatory.

Table 3–9 lists the input parameters you set in the OrderNote container object. Each parameter in the table indicates whether it is optional or mandatory in the container object for this API.

Table 3–9 OrderNote Input Parameters

Input Parameter	Parameter Information
noteId	**Data Type:** int **Description:** Note ID for the update. This parameter is mandatory.
note	**Data Type:** String **Description:** Note information. This parameter is mandatory.
dateEntered	**Data Type:** java.util.Calendar **Description:** Date of the note entered. This parameter is set by the business logic and if provided is ignored. This is for internal use only.

Table 3–9 (Cont.) OrderNote Input Parameters

Input Parameter	Parameter Information
orderKey	**Data Type:** com.metasolv.value.EntityKey **Description:** Order key of the note. This parameter is not needed as input because the noteId is used to retrieve the note. This is for internal use only.
noteNumber	**Data Type:** String **Description:** A string reference of the note. This parameter is optional.
systemGeneratedInd	**Data Type:** String **Description:** System indicator of the note. This is for internal use only. This parameter is internal.
userId	**Data Type:** String **Description:** Userid who updated the note. This parameter is optional.

Sample Data for Input Parameters

Example 3–4 is an example of the data you must set for the input parameters to update an Engineering Work Order note.

Example 3–4 Data Values for Input Parameters

```
protected static ValueFactory valueFact;
// create the value factory
valueFact = GenericFactory.makeValueFactory();
// create the order note object using the value factory and setting the
// information on it.
OrderNote orderNote = (OrderNote) valueFact.makeInstance(OrderNote.class);
orderNote.setNoteId(40124);
orderNote.setNote("Test EWO Notes");
```

Associating a Connection to a Work Order (associateConnectionToWorkOrder)

This API supports association of a connection to an Engineering Work Order based on the input data provided.

EJB API Call

```
WorkOrderManagerRemote.associateConnectionToWorkOrder (
   User user,
   AssociateConnectionToWorkOrderPolicy policy)
```

Container Object

```
com.metasolv.value.order.AssociateConnectionToWorkOrderPolicy
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns the successfully processed connections.

- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 3–10 shows the input parameters for the API.

Table 3–10 Input Parameters for the API

Input Parameter	Parameter Information
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.
policy	**Data Type:** com.metasolv.value.order.AssociateConnectionToWorkOrderPolicy **Description:** Contains information to associate a connection to an Engineering Work Order. This parameter is mandatory.

Table 3–11 lists the input parameters you set in the AssociateConnectionToWorkOrderPolicy container object. Each parameter in the table indicates whether it is optional or mandatory in the container object.

Table 3–11 AssociateConnectionToWorkOrderPolicy Input Parameters

Input Parameter	Parameter Information
activityCd	**Data Type:** com.metasolv.value.order.AssociateConnectionToWorkOrderActivityCd **Description:** Activity code for the connection on the work order. This is an enumeration with the following values: ■ NEW ■ CHANGE ■ DISCONNECT ■ CANCELED This parameter is mandatory.
connectionKey	**Data Type:** com.metasolv.value.resource.entity.connection.ConnectionKey **Description:** Key of the connection to be added to the work order. This field is used in the validation to ensure the connection is not already associated to the order. This parameter is mandatory.
workOrderKey	**Data Type:** com.metasolv.value.order.WorkOrderKey **Description:** Key of the work order to which connection must be added. The order must not be completed or cancelled. This parameter is mandatory.

Table 3–11 (Cont.) AssociateConnectionToWorkOrderPolicy Input Parameters

Input Parameter	Parameter Information
mpoSupported	**Data Type:** boolean **Description:** Sets multiple pending order supported indicator. This parameter is optional.
validateOnly	**Data Type:** boolean **Description:** Sets validate only indicator. When this is true, only the validation logic runs for the connections and no associations are persisted. This validates the following: ■ The connectionKey is populated ■ The connection is not already on the work order ■ The activity code is provided ■ There are no assignments if it is a disconnect order If the multiple pending order supported parameter is set, then the connection cannot be on any other **open** work order. This parameter is optional.
implementationContext	**Data Type:** com.metasolv.value.ImplementationContext **Description:** Sets the implementation context. This parameter is optional.

Sample Data for Input Parameters

Example 3–5 is an example of the data you set for the input parameters to associate a connection to an existing Engineering Work Order.

Example 3–5 Data Values for Input Parameters

```
protected static ValueFactory valueFact;
private ValidValueAccessManager validValueAccessManager;

// create the value factory
valueFact = GenericFactory.makeValueFactory();

// create the valid value access manager
validValueAccessManager = GenericFactory.makeValidValueAccessManager(userContext);

// creating the AssociateConnectionToWorkOrderPolicy object using the value
factory.
AssociateConnectionToWorkOrderPolicy assocConnToWoPolicy = (
   AssociateConnectionToWorkOrderPolicy)valueFact.makeInstance(
   AssociateConnectionToWorkOrderPolicy.class);

// creating the work order object using the value factory and
// setting order number in and setting it to the policy.
WorkOrderKey woKey = (WorkOrderKey)valueFact.makeInstance(WorkOrderKey.class);
woKey.setDocumentNumber(408792);
assocConnToWoPolicy.setWorkOrderKey(woKey);

// creating the connection object using the value factory and
// setting order number in and setting it to the policy.
ConnectionKey connKey =
(ConnectionKey)valueFact.makeInstance(ConnectionKey.class);
connKey.setConnectionId(2047281);
assocConnToWoPolicy.setConnectionKey(connKey);
```

```
// specify the MOP supported or not
assocConnToWoPolicy.setMpoSupported(true);

// creating the connection activity code and assigning the same to the policy.
AssociateConnectionToWorkOrderActivityCd assocConnToWoActCd =
   validValueAccessManager.getValue(
      AssociateConnectionToWorkOrderActivityCd.TYPE,
      AssociateConnectionToWorkOrderActivityCd.NEW);

assocConnToWoPolicy.setActivityCd(assocConnToWoActCd);
```

Associating Equipment to a Work Order (associateEquipmentToWorkOrder)

This API supports association of equipment to an Engineering Work Order based on the input data provided.

EJB API Call

```
WorkOrderManagerRemote.associateEquipmentToWorkOrder (
   User user,
   AssociateEquipmentToWorkOrderPolicy policy)
```

Container Object

```
com.metasolv.value.order.AssociateEquipmentToWorkOrderPolicy
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns the array list of successfully associated equipment.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 3–12 shows the input parameters for the API.

Table 3–12 Input Parameters for the API

Input Parameter	Parameter Information
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.
policy	**Data Type:** com.metasolv.value.order.AssociateEquipmentToWorkOrderPolicy **Description:** Contains information to associate equipment to an Engineering Work Order. This parameter is mandatory.

Table 3–13 lists the objects you need to set within the AssociateEquipmentToWorkOrderPolicy container object.

Table 3–13 Containers To Set Within the AssociateEquipmentToWorkOrderPolicy Container Object

Input Parameter	Data Type
workOrderKey	com.metasolv.value.EntityKey
equipmentKey	com.metasolv.value.EntityKey
parentEquipmentKey	com.metasolv.value.EntityKey
childEquipmentKey	com.metasolv.value.EntityKey
activityCd	com.metasolv.value.order.AssociateConnectionToWorkOrderActivityCd
implementationContext	com.metasolv.value.ImplementationContext

Table 3–14 lists the input parameters you set in the AssociateEquipmentToWorkOrderPolicy container object. Each parameter in the table indicates whether it is optional or mandatory in the container object.

Table 3–14 AssociateEquipmentToWorkOrderPolicy Input Parameters

Input Parameter	Parameter Information
activityCd	**Data Type:** com.metasolv.value.order.AssociateConnectionToWorkOrderActivityCd **Description:** Activity code for the equipment on the work order. This parameter is mandatory.
equipmentKey	**Data Type:** com.metasolv.value.EntityKey **Description:** Key of the equipment to be added to the work order. This parameter is mandatory.
parentEquipmentKey	**Data Type:** com.metasolv.value.EntityKey **Description:** Key of the equipment to be added to the work order. This parameter is optional.
childEquipmentKey	**Data Type:** com.metasolv.value.EntityKey **Description:** Key of the equipment to be added to the work order. This parameter is optional.
workOrderKey	**Data Type:** com.metasolv.value.EntityKey **Description:** Key of the work order to which equipment must be added. This parameter is mandatory.

Table 3–14 (Cont.) AssociateEquipmentToWorkOrderPolicy Input Parameters

Input Parameter	Parameter Information
mpoSupported	**Data Type:** boolean **Description:** Sets multiple pending order supported indicator. This parameter is optional.
validateOnly	**Data Type:** boolean **Description:** Sets validate only indicator. When this is true, only the validation logic runs and no associations are persisted. This validates the following: ■ The work order key is valid ■ The work order is not complete or cancelled ■ The parent and child equipment keys are valid, and the parent equipment key is not in a "Spare" status ■ The child equipment keys are only populated when the caller does a copy operation (Refer to the Javadoc for more information) ■ The activity code is populated and valid This parameter is optional.
implementationContext	**Data Type:** com.metasolv.value.ImplementationContext **Description:** Sets the implementation context. This parameter is optional.

Sample Data for Input Parameters

Example 3–6 is an example of the data you set for the input parameters to associate equipment to an existing Engineering Work Order.

Example 3–6 Data Values for Input Parameters

```
protected static ValueFactory valueFact;
private ValidValueAccessManager validValueAccessManager;

// create the value factory
valueFact = GenericFactory.makeValueFactory();

// create the valid value access manager
validValueAccessManager = GenericFactory.makeValidValueAccessManager(userContext);

// Creating the AssociateEquipmentToWorkOrderPolicy object using the value
factory.
AssociateEquipmentToWorkOrderPolicy assocEquipToWoPolicy = (
   AssociateEquipmentToWorkOrderPolicy)valueFact.makeInstance(
      AssociateEquipmentToWorkOrderPolicy.class);

// Creating the work order object using the value factory and
// setting order number in and setting it to the policy.
WorkOrderKey woKey = (WorkOrderKey)valueFact.makeInstance(WorkOrderKey.class);
woKey.setDocumentNumber(408792);
assocEquipToWoPolicy.setWorkOrderKey(woKey);

// Creating the equipment object using the value factory and
// setting order number in and setting it to the policy.
EquipmentKey equipKey = (EquipmentKey)valueFact.makeInstance(EquipmentKey.class);
equipKey.setEquipmentId(331534);
assocEquipToWoPolicy.setEquipmentKey(equipKey);
```

```
// Creating the equipment activity code and assigning the same to the policy.
AssociateEquipmentToWorkOrderActivityCd assocEquipToWoActCd =
   validValueAccessManager.getValue(
      AssociateEquipmentToWorkOrderActivityCd.TYPE,
      AssociateEquipmentToWorkOrderActivityCd.NEW_INSTALL);
assocEquipToWoPolicy.setActivityCd(assocEquipToWoActCd);
```

Adding a Task to a Work Order (addTask)

This API supports assign a task to an Engineering Work Order based on the input data provided.

EJB API Call

```
WorkManagerRemote.addTask (
   Task task,
   TaskKey taskKey,
   AddTaskPolicy policy,
   User user)
```

Container Object

```
com.metasolv.value.work.Task
```

```
com.metasolv.value.work.TaskKey
```

```
com.metasolv.value.work.AddTaskPolicy
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns the array list of successfully processed connections.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 3–15 shows the input parameters for the API.

Table 3–15 Input Parameters for the API

Input Parameter	Parameter Information
task	**Data Type:** com.metasolv.value.work.Task **Description:** Contains information of a task to associate an Engineering Work Order. This parameter is mandatory.

Table 3–15 (Cont.) Input Parameters for the API

Input Parameter	Parameter Information
taskKey	**Data Type:** com.metasolv.value.work.TaskKey **Description:** Contains information of a task key to associate an Engineering Work Order. This parameter is mandatory.
policy	**Data Type:** com.metasolv.value.work.AddTaskPolicy **Description:** Contains information of the task policy to associate to an Engineering Work Order. This parameter is mandatory.
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.

Table 3–16 lists the input parameters you set in the AddTaskPolicy container object. Each parameter in the table indicates whether it is optional or mandatory in the container object.

Table 3–16 AddTaskPolicy Input Parameters

Input Parameter	Parameter Information
addTaskPosition	**Data Type:** com.metasolv.value.work.AddTaskPosition **Description:** Contains the information of task position. This parameter is mandatory.
returnWorkPlan	**Data Type:** boolean **Description:** Work plan required or not as part of the return object. This parameter is mandatory.
returnWorkPlanDefinition	**Data Type:** boolean **Description:** Work plan definition required or not as part of the return object. This parameter is optional.
returnTaskList	**Data Type:** boolean **Description:** Task list required or not as part of the return object. This parameter is optional.
processRulesAndBehaviors	**Data Type:** boolean **Description:** Flag to determine if rules and behaviors is processed or not. This parameter is optional.

Table 3–17 lists the object you need to set within the TaskKey container object.

Table 3–17 TaskKey Input Parameters

Input Parameter	Parameter Information
orderKey	**Data Type:** com.metasolv.value.order.OrderKey **Description:** Sets the order value. This parameter is mandatory.
taskId	**Data Type:** int **Description:** Contains the value of the task ID which is the identifier for the task. This parameter is mandatory.

Table 3–18 lists the objects you must set within the Task container object.

Table 3–18 Containers To Set Within the Task Container Object

Input Parameter	Data Type
orderKey	com.metasolv.value.order.OrderKey
taskStatus	com.metasolv.value.work.TaskStatus
taskCriticalDateIndicator	com.metasolv.value.YesNoIndicator
autoCompInd	com.metasolv.value.YesNoIndicator
rejectStatus	com.metasolv.value.YesNoIndicator
queueStatus	com.metasolv.value.work.QueueStatus
smartTaskInd	com.metasolv.value.YesNoIndicator

Table 3–19 lists the input parameters you set in the Task container object. Each parameter in the table indicates whether it is optional or mandatory in the container object.

Table 3–19 Task Input Parameters

Input Parameter	Parameter Information
orderKey	**Data Type:** com.metasolv.value.order.OrderKey **Description:** Contains the order number. This parameter is mandatory.
taskId	**Data Type:** int **Description:** Contains the value of the task ID which is the identifier for the task. This parameter is mandatory.
taskType	**Data Type:** String **Description:** Contains the task type. This parameter is optional.
taskStatus	**Data Type:** com.metasolv.value.work.TaskStatus **Description:** Contains the task status. This parameter is optional.
scheduledCompletionDate	**Data Type:** java.util.Calendar **Description:** Contains the scheduled completion date. This parameter is optional.

Table 3–19 (Cont.) Task Input Parameters

Input Parameter	Parameter Information
actualReleaseDate	**Data Type:** java.util.Calendar **Description:** Contains the actual release date. This parameter is optional.
revisedCompletionDate	**Data Type:** java.util.Calendar **Description:** Contains the revised completion date. This parameter is optional.
estimatedCompletionDate	**Data Type:** java.util.Calendar **Description:** Contains the estimated completion date. This parameter is optional.
workQueueId	**Data Type:** String **Description:** Contains the work queue name. This parameter is optional.
actualCompletionDate	**Data Type:** java.util.Calendar **Description:** Contains the actual completion date. This parameter is optional.
taskCriticalDateIndicator	**Data Type:** boolean **Description:** Contains the task critical date indicator. This parameter is optional.
taskStatusDate	**Data Type:** java.util.Calendar **Description:** Contains the task status date. This parameter is optional.
firstJeopardyId	**Data Type:** int **Description:** Contains the first jeopardy id. This parameter is optional.
autoCompInd	**Data Type:** com.metasolv.value.YesNoIndicator **Description:** Contains auto completion indicator. This parameter is optional.
rejectStatus	**Data Type:** com.metasolv.value.YesNoIndicator **Description:** Reflects the task reject status. This parameter is optional.
taskPriority	**Data Type:** int **Description:** Contains the priority of the task. This parameter is optional.
circuitDesignId	**Data Type:** int **Description:** Contains the circuit design ID assigned to the task. This parameter is optional.
workQueuePriority	**Data Type:** int **Description:** Contains the work queue priority. This parameter is optional.

Table 3–19 (Cont.) Task Input Parameters

Input Parameter	Parameter Information
assignedFromWorkQueue	**Data Type:** String **Description:** Indicates whether this task is assigned from the work queue or not. This parameter is optional.
assignedFromDate	**Data Type:** java.util.Calendar **Description:** Contains the assigned from date. This parameter is optional.
queueStatus	**Data Type:** com.metasolv.value.work.QueueStatus **Description:** Contains the queue status. This parameter is optional.
billingStatus	**Data Type:** char **Description:** Contains the billing status. This parameter is optional.
scheduledReleaseDate	**Data Type:** java.util.Calendar **Description:** Contains the scheduled release date. This parameter is optional.
sortPriority	**Data Type:** String **Description:** Contains the sort priority. This parameter is optional.
requiredInd	**Data Type:** char **Description:** Contains the indicator that informs you whether this task is required or not. This parameter is optional.
systemGenInd	**Data Type:** char **Description:** Contains the indicator that informs you whether this task is system generated or not. This parameter is optional.
reqPlanId	**Data Type:** int **Description:** Contains the requested plan id. This parameter is optional.
taskOpenInd	**Data Type:** char **Description:** Contains the indicator that informs you whether this task is opened or not. This parameter is optional.
jobId	**Data Type:** int **Description:** Contains the job identifier. This parameter is optional.
sequence	**Data Type:** int **Description:** Contains the task sequence. This parameter is optional.

Table 3–19 (Cont.) Task Input Parameters

Input Parameter	Parameter Information
taskPrompt	**Data Type:** char **Description:** Contains the task prompt. This parameter is optional.
assignDtCd	**Data Type:** char **Description:** Contains the assigned date code. This parameter is optional.
servItemId	**Data Type:** int **Description:** Contains the service item id assigned to the task. This parameter is optional.
completionDays	**Data Type:** int **Description:** Contains the days taken for the task completion. This parameter is optional.
completionHours	**Data Type:** int **Description:** Contains the hours taken for the task completion. This parameter is optional.
completionMinutes	**Data Type:** int **Description:** Contains the minutes taken for the task completion. This parameter is optional.
potentiallyLateDays	**Data Type:** int **Description:** Contains the number of days that a task is potentially late. This parameter is optional.
potentialLateHours	**Data Type:** int **Description:** Contains the number of hours that a task is potentially late. This parameter is optional.
potentialLateMinutes	**Data Type:** int **Description:** Contains the number of minutes that a task is potentially late. This parameter is optional.
closeOfBusinessInd	**Data Type:** char **Description:** Contains the close of day business indicator. This parameter is optional.
latePromptInd	**Data Type:** char **Description:** Contains the late prompt indicator. This parameter is optional.
systemTaskInd	**Data Type:** char **Description:** Indicates whether this task is a system task or not. This parameter is optional.
taskLabel	**Data Type:** String **Description:** Contains the label of the task. This parameter is optional.

Table 3–19 (Cont.) Task Input Parameters

Input Parameter	Parameter Information
executionPoint	**Data Type:** char **Description:** Contains the execution point of the task. This parameter is optional.
dispositionDays	**Data Type:** int **Description:** Contains the number of disposition days. This parameter is optional.
prevWorkQueueId	**Data Type:** String **Description:** Contains the previous work queue identifier name. This parameter is optional.
taskOpenedInd	**Data Type:** char **Description:** Indicates whether the task is opened or not. This parameter is optional.
orderedFacilityIndicator	**Data Type:** char **Description:** Contains the ordered facility indicator. This parameter is optional.
taskTypeDescription	**Data Type:** String **Description:** Contains the task type description. This parameter is optional.
transactionOrigin	**Data Type:** String **Description:** Contains the transaction origin. This parameter is optional.
customerViewable	**Data Type:** char **Description:** Indicates whether this task is viewable by the customer or not. This parameter is optional.
relatedArea	**Data Type:** String **Description:** Contains the related area of this task. This parameter is optional.
appointmentTask	**Data Type:** char **Description:** Indicates whether this task is an appointment or not. This parameter is optional.
graceDays	**Data Type:** int **Description:** Contains the grace days for this task. This parameter is optional.
exeFileNm	**Data Type:** String **Description:** Contains the executable file name for this task. This parameter is optional.
graceMinutes	**Data Type:** int **Description:** Contains the number of grace minutes for this task. This parameter is optional.

Table 3–19 (Cont.) Task Input Parameters

Input Parameter	Parameter Information
graceHours	**Data Type:** int **Description:** Contains the number of grace hours for this task. This parameter is optional.
smartTaskInd	**Data Type:** com.metasolv.value.YesNoIndicator **Description:** Indicates whether this task is a smart task or not. This parameter is optional.
validationRequired	**Data Type:** boolean **Description:** Indicates whether any validations are performed for this task or not. This parameter is optional.
windowNm	**Data Type:** String **Description:** Contains the window name for this task. This parameter is optional.
combinSmartSysInd	**Data Type:** char **Description:** Indicates whether the smart task and the system task can be combined or not. This parameter is optional.
toolNm	**Data Type:** String **Description:** Contains the tool name for this task. This parameter is optional.
hasAccess **Note**: This parameter is not used in the web service operation.	**Data Type:** boolean **Description:** Indicates whether this task has access or not. This parameter is optional.

Sample Data for Input Parameters

Example 3–7 is an example of the data you set for the input parameters to associate a task to a existing Engineering Work Order.

Example 3–7 Data Values for Input Parameters

```
protected static ValueFactory valueFact;
private ValidValueAccessManager validValueAccessManager;

// Create the value factory
valueFact = GenericFactory.makeValueFactory();

// Create the valid value access manager
validValueAccessManager =
   GenericFactory.makeValidValueAccessManager(userContext);

// Creating the WorkPlanAssignmentPolicy object using the value factory and
// setting the values in it.
WorkPlanAssignmentPolicy workPlanAssignPolicy = (WorkPlanAssignmentPolicy)
   valueFactory.makeInstance(WorkPlanAssignmentPolicy.class);
workPlanAssignPolicy.setReturnTaskList(true);
workPlanAssignPolicy.setReturnWorkPlan(true);
workPlanAssignPolicy.setReturnWorkPlanDefinition(true);
workPlanAssignPolicy.setProcessRulesAndBehaviors(true);
```

```
// Creating the work WorkPlanDefinitionKey object using the
// value factory and setting the values in it.
WorkPlanDefinitionKey workPlanDefKey= (WorkPlanDefinitionKey)
   valueFactory.makeInstance(WorkPlanDefinitionKey.class);
workPlanDefKey.setWorkPlanDefinitionId(28917);

// Creating the order key object using the value factory and setting the
// values in it.
OrderKey orderKey = (OrderKey)valueFactory.makeInstance(OrderKey.class);
orderKey.setDocumentNumber(14549413);

// Creating the task object and setting the values in it.
Task taskToInsert= (Task) valueFactory.makeInstance(Task.class);
taskToInsert.setCompletionDays(1);
taskToInsert.setTaskType("APP");
taskToInsert.setWorkQueueId("ISR_CORD");
// Creating the task key object and setting the values in it.
TaskKey taskKey= (TaskKey) valueFactory.makeInstance(TaskKey.class);
taskKey.setTaskId(54868178);
taskKey.setOrderKey(orderKey);

// Creating the add task policy object and setting the values in it.
AddTaskPolicy taskPolicy= (AddTaskPolicy)
   valueFactory.makeInstance(AddTaskPolicy.class);
taskPolicy.setReturnTaskList(true);
taskPolicy.setReturnWorkPlan(true);
taskPolicy.setReturnWorkPlanDefinition(true);
taskPolicy.setAddTaskPosition(validValueAccessManager.getValue(
   AddTaskPosition.TYPE , 5));
```

Supplementing a Work Order (processDDChangeSupplement)

This API supports process supplementing of an Engineering Work Order based on the input data provided.

EJB API Call

```
WorkOrderManagerRemote.processDDChangeSupplement (
   User user,
   WorkOrder workOrder,
   String suppNote)
```

Container Object

```
com.metasolv.value.order.WorkOrder
```

Return Object Type from API

com.mslv.ejb.EJBReturn

To obtain the API output, use the following methods in the return object:

- getReturnObject(): Returns the processed work order.
- getMessages(): Returns a list of error messages in the form of a vector.

Input Parameters for the API

Table 3–20 shows the input parameters for the API.

Table 3–20 Input Parameters for the API

Input Parameter	Parameter Information
user	**Data Type:** com.mslv.ejb.User **Description:** Contains user information. This parameter is mandatory.
workOrder	**Data Type:** com.metasolv.value.order.WorkOrder **Description:** Contains information to supplement an Engineering Work Order. This parameter is mandatory.
suppNote	**Data Type:** String **Description:** Contains the supplement note information. This parameter is mandatory.

Table 3–21 lists the input parameters you set in the WorkOrder container object. Each parameter in the table indicates whether it is optional or mandatory in the container object.

Table 3–21 WorkOrder Input Parameters

Input Parameter	Parameter Information
documentNumber	**Data Type:** int **Description:** Document number of the order. This field must be provided only when updating an existing order. For new orders this is going to be created automatically. This parameter is mandatory.
description	**Data Type:** String **Description:** Description provided for the order. This parameter is optional.
dueDate	**Data Type:** java.util.Calendar **Description:** Due date of the order. If this is provided then it cannot be in the past. This parameter is optional. If this parameter is not provided, this field will be set to the current date.
projectKey **Note**: This parameter is obsolete.	**Data Type:** com.metasolv.value.EntityKey **Description:** Project ID of the order. This parameter is optional.
referenceId	**Data Type:** String **Description:** Reference ID of the order. This parameter is optional.
responsiblePerson	**Data Type:** String **Description:** Responsible person name of the order. This parameter is optional.
status	**Data Type:** com.metasolv.value.WorkOrderStatus **Description:** Status of the order. This parameter is mandatory.

Table 3–21 (Cont.) WorkOrder Input Parameters

Input Parameter	Parameter Information
supplementType	**Data Type:** com.metasolv.value.order.WorkOrderSupplementType **Description:** Supplement type of the order. This field must be provided only when updating an existing order by supplementing it. This should not be set for the new orders. This parameter is optional.
orderNumber	**Data Type:** String **Description:** The order number. If this parameter is populated, it must be unique, for example, no other EWOs exist with the same work order number. Also, the length must not exceed 17 characters. This parameter is optional.
isAutoWorkOrder **Note**: This parameter is obsolete.	**Data Type:** boolean **Description:** This parameter is obsolete and must be set to **false**. This parameter is optional.

Sample Data for Input Parameters

Example 3–8 is an example of the data you set for the input parameters to process supplement an existing Engineering Work Order.

Example 3–8 Data Values for Input Parameters

```
protected static ValueFactory valueFact;
private ValidValueAccessManager validValueAccessManager;

// create the value factory
valueFact = GenericFactory.makeValueFactory();

// create the valid value access manager
validValueAccessManager = GenericFactory.makeValidValueAccessManager(userContext);

// create the work order object using the value factory.
WorkOrder wo = (WorkOrder)valueFact.makeInstance(WorkOrder.class);

// setting the document number.
wo.setDocumentNumber(638178);

// create the status object using the value factory and
// setting the values in it.
WorkOrderStatus woStatus = validValueAccessManager.getValue(
   WorkOrderStatus.TYPE,WorkOrderStatus.IN_PROGRESS);
wo.setStatus(woStatus);

// create the supplement type object using the value factory and
// setting the values in it.
WorkOrderSupplementType wost =
   validValueAccessManager.getValue(WorkOrderSupplementType.TYPE, "2");
wo.setSupplementType(wost);
```