# [IMS Request 1680344](https://dev.azure.com/AVEVA-VSTS/Issue%20Management%20System/_workitems/edit/1680344%22%20%5Ct%20%22_blank): RemConnJoin edit causes GRT specific 'emprise' field content removed

Contents

[IMS Request 1680344: RemConnJoin edit causes GRT specific 'emprise' field content removed 1](#_Toc92866541)

[Summary of IMS Request 1](#_Toc92866542)

[Background of ADE Functionality 1](#_Toc92866543)

[Syntax of Assignconn Fnput 1](#_Toc92866544)

[Syntax of Unassignconn Fnput 2](#_Toc92866545)

[Suggested Solutions 3](#_Toc92866546)

[Custom Fnput or BLT Wrapper 3](#_Toc92866547)

[Virtual Field 3](#_Toc92866548)

# Summary of IMS Request

GRT has added a custom field, “emprise”, to the RemConnJoin table. The field contents can be loaded via DBLL.

As soon as any edit is done to any field in the remconnjoin record from ADE and the changes are saved, the existing content of this “emprise” field is removed (lost).

# Background of ADE Functionality

ADE issues BLT Fnputs against the remote table to create, “update”, or delete remconnjoin records in RealTime.

* To create a remconnjoin record, it issues an “assignconn” Fnput against the remote, passing in a hard-coded list of field values into the “assignconn” Fnput from the ADE custom form. The “assignconn” Fnput then updates that list of field values into the newly created remconnjoin record.
* To delete a remconnjoin record, it issues an “unassignconn” Fnput against the remote.
* To “update” a remconnjoin record:
	+ It first issues an “assignconn” Fnput against the remote to assign the new connection, creating a new remconnjoin record.
	+ It then issues an “unassignconn” Fnput against the remote to delete the pre-existing remconnjoin record.

## Syntax of Assignconn Fnput

The following shows the syntax required to call the “assignconn” fnput:

 List<DALBLTParam> bltParameters = new List<DALBLTParam>();

 List<DALBLTParam> bltValues = new List<DALBLTParam>();

 Object clientObject = new object();

 bltParameters.Add(new DALBLTParam("remote", false));

 bltParameters.Add(new DALBLTParam(remoteName, false));

 // Build the command to run

 StringBuilder commandToRun = new StringBuilder();

 commandToRun.Append("assignconn ");

 commandToRun.Append(connectionName);

 commandToRun.Append(" ");

 commandToRun.Append(costFactor);

 commandToRun.Append(" ");

 commandToRun.Append(failureTimeout);

 commandToRun.Append(" ");

 commandToRun.Append(inhAutoPoll);

 commandToRun.Append(" ");

 commandToRun.Append(goodTestFreq);

 commandToRun.Append(" ");

 commandToRun.Append(badTestFreq);

 // Add the command to run to the BLT call

 bltParameters.Add(new DALBLTParam(commandToRun.ToString(), false));

 DALBLTConnection bltConnection = DALBLTClient.GetGlobalConnection(

 "CMX",

 null);

 int rowsAffected;

 DataSet resultsDS;

 DALBLTExecutionStatus execStatus;

 string failureReason;

 //Explicitly defined the displayName to avoid permission issue with FQDN

 bltConnection.DisplayName = DNAEnvironment.PhysicalDnsMachineName;

 DebugLogger.Write(DebugLogger.LogLevel.Verbose,

 "About to execute BLT call fnput");

 if (!bltConnection.ExecuteSync(

 "fnput",

 0, // Unlimited rows back

 bltParameters,

 out bltValues,

 out rowsAffected,

 out resultsDS,

 out execStatus,

 out failureReason))

## Syntax of Unassignconn Fnput

The following shows the syntax required to call the “unassignconn” fnput:

 List<DALBLTParam> bltParameters = new List<DALBLTParam>();

 List<DALBLTParam> bltValues = new List<DALBLTParam>();

 Object clientObject = new object();

 bltParameters.Add(new DALBLTParam("remote", false));

 bltParameters.Add(new DALBLTParam(remoteName, false));

 // Build the command to run (fnput remote.0remote "unassignconn test1")

 StringBuilder commandToRun = new StringBuilder();

 commandToRun.Append("unassignconn ");

 commandToRun.Append(connectionName);

 // Add the command to run to the BLT call

 bltParameters.Add(new DALBLTParam(commandToRun.ToString(), false));

 DALBLTConnection bltConnection = DALBLTClient.GetGlobalConnection(

 "CMX",

 null);

 int rowsAffected;

 DataSet resultsDS;

 DALBLTExecutionStatus execStatus;

 string failureReason;

 //Explicitly defined the displayName to avoid permission issue with FQDN

 bltConnection.DisplayName = DNAEnvironment.PhysicalDnsMachineName;

 DebugLogger.Write(DebugLogger.LogLevel.Verbose,

 "About to execute BLT call fnput");

 if (!bltConnection.ExecuteSync(

 "fnput",

 0, // Unlimited rows back

 bltParameters,

 out bltValues,

 out rowsAffected,

 out resultsDS,

 out execStatus,

 out failureReason))

# Suggested Solutions

## Custom Fnput or BLT Wrapper

One suggested work around is to create a GRT custom “assignconn” Fnput or BLT. This new custom Fnput/BLT would:

* First call the assignconn fnput as illustrated above to create the new remconnjoin record
* Update the remconnjoin record “emprise” field after successful creation of the remconnjoin record.

This custom Fnput/BLT would then be called from a custom ADE form created by GRT.

## Virtual Field

Depending upon the use case required for the “emprise” field in the remconnjoin record, the “emprise” field could be created as a virtual field. Access to the field’s value would only apply if done using the VDB layer or something that depends on the VDB layer (dbget, v\_fldget, RealTimeVPath.FieldGet, etc). It would not be available if access is done through the set layer/HPDBset layer, or any code at any layer that directly accesses shared memory using a locked record.

For example, you would **not** be able to:

 table lock (remconnjoin)

 record lock (remconnjoinRecord)

 empriseValue = remconnjoinRecord.emprise

A fldget override could then be created such that any application doing a fldget against the “emprise” field would access the virtual fldget override against the “emprise” field, which would then perform a lookup against the remote record to retrieve the value.