

[0253] Sub-Elements

[0254] There are no sub-elements.

[0255] Parameter & Column Binding Semantics

[0256] The Default Binding feature is intended to provide a syntactical shortcut to users so that when the names of the Custom Table column match with the underlying parameter and/or result column name then the system behaves as if the binding is present as a Parameter name=CustomColumn name binding or ResultColumn name=CustomColumn name binding. The Default Binding feature does not in any way alter the semantics of what it means for a parameter or column binding to be present (or absent) in any given scenario, but is simply a mechanism for attempting to infer the appropriate bindings. Default binding is on by default and can be disabled per command.

[0257] Default Binding with Explicit Metadata. When there is explicit metadata (compile-time information) for how many parameters or result columns should be expecting for a given command, the exact number of default bindings can be created up to the number of parameters or columns defined on the procedure or routine. The following table summarizes which command types for which explicit compile-time parameter/column information exists.

	Parameters	Column
User-Defined Functions	Yes	Yes
Stored Procedures	Yes	No
Inline SQL	No	No

[0258] When Default Binding is turned on, the mapping framework creates a binding for each column in the custom table to the appropriately named parameter and/or column in the command. When metadata is available, an explicit binding is assumed to be a partial binding, and the system will attempt to apply default binding to any remaining columns on the Custom table. For example, consider the CustomTable for Orders which is defined as (oid nchar(10), odate date, comments nvarchar(max)). The InsertCommand for the Orders table has been overridden to reference sp_OrderInsert(@orderid nchar(10), odate date, comments nvarchar(max)). With Default Binding turned on and no ParameterBindings explicitly specified, the mapping framework would generate two bindings: one for odate and one for comments. The user could then specify an explicit binding from oid to orderid so that all three parameters are bound.

[0259] Default Binding without Explicit Metadata. In cases where no declarative information about parameters and/or columns is available, the system will generate a parameter and a column binding for each column in the custom table. This may result in over-binding where there are more custom table columns than parameters in a procedure or columns in a result set.

[0260] Partial Binding is still supported in this scenario, so in every case where Default Binding has not been disabled there will be one parameter binding and one column binding for each column in the custom table.

[0261] The user can turn-off Default Binding explicitly and specify all of the bindings explicitly.

[0262] Binding Semantics

[0263] The expected behavior is capable of being defined when the custom table has more or less bindings than the input parameters to a procedure or the result columns to output parameters for a particular command. Where the CustomTable is wider than the Query Result Set, this scenario is designed to accommodate write-only fields (e.g., time of update, or userid doing the update). The mapped columns on the CustomTable that are referenced by a FieldMap, but not contained in the ResultSet, are treated as if the return is null. The target domain defines what the appropriate behavior is (null behavior for XML and default value form constructor in Objects).

[0264] UDF Scenario

[0265] In a UDF scenario, the appropriate number of columns is always selected from the UDF result set because compile time metadata allows formulation of queries properly. If the custom table is still wider than the UDF result set, the null behavior specified above will be executed.

[0266] Inline SQL Scenario

[0267] In this scenario if the user has disabled default binding and properly listed the columns to be returned from the SQL statement, the behavior is identical to the UDF scenario. If the user is relying on default bindings and the custom table is wider than the result set, columns are selected that do not exist resulting in a runtime error.

[0268] CustomTable Narrower than Query Result Set

[0269] In this scenario, the user is accessing a UDF or StoredProcedure that cannot be altered, but which returns values which are not being used in the customer's mapping application. This is not an error case, and the unbound/unmapped values are ignored.

[0270] This scenario is unlikely in the Inline case because only the result columns that are bound to custom table columns will be selected.

[0271] CustomTable Wider than Input Parameter Set for CUD

[0272] In the update case, Custom columns that are not bound to columns or parameters, are ignored. It is assumed that these are read-only fields and/or not required for execution of the command (particularly true for Delete commands which may only take the key).

[0273] In the Stored Procedure case, the number of parameters is always known and declared so over-binding never occurs. A compile time error can be provided if the user attempts to explicitly bind to a parameter that does not exist.

[0274] In the Inline SQL scenario, over-binding of the number of parameters that is used in the command may occur. This is not an error and may simply result in extra parameter values being sent across the wire.

[0275] CustomTable Narrower than Parameter Set

[0276] In this case the Command requires more parameters than are currently mapped from the user's application. If the Command has parameters that are not bound to any column on the Custom Table, or the Column on the Custom Table is not referenced by a FieldMap we will always pass