

<u>Sub-Elements</u>		
Tag	Card	Comment
PrimaryKey	0-1	A table may have a primary key. At most one is allowed. Primary key can be defined on a set of columns. The PrimaryKey element will refer to these columns.
AlternateIdstructure	0-1	This is a logical (user-supplied) element which defines a set of columns which uniquely identify a row in the table without using that table's primary key

**[0190]** 31. PrimaryKey

**[0191]** This element represents a Primary Key for a table. It is always defined in the context of a Table. It references the column(s) that define a primary key for this table.

Tag	Card	Comment
<u>Attributes</u>		
Name	Req	Constraint name.
<u>Sub-Elements</u>		
ColumnRef	1+	Each ColumnRef refers to a column that is a part of this primary key.

**[0192]** 32. AlternateIdentity,

**[0193]** AlternateIdentity is used by the BulkLoad upsert feature. The columns defined by the AlternateIdentity constraint can be used to uniquely identify a row in a table without making use of the PrimaryKey.

<u>Sub-Elements</u>		
Tag	Card	Comment
ColumnRef	1+	Each ColumnRef identifies a single column on the current table that is part of the AlternateIdstructure for that table

**[0194]** RSD Generation Notes

**[0195]** AlternateIdentity is logical by default and must be preserved by RSD Generators when they refresh from the database.

**[0196]** 33. ColumnRef

**[0197]** This element is used to refer to a column previously defined, and is used to refer to columns from constraints, etc.

<u>Attributes</u>		
Tag	Card	Comment
Name	Req	The name of a column that belongs to the parent Table that contains the PrimaryKey element.

**[0198]** Custom Tables

**[0199]** The Custom Tables feature is a mechanism whereby the user can support database operations as a means to make up a logical table. The user can perform transformations on the physical data when reading or writing from the database. Custom Tables enables the capability to map fields from the target domain to Commands (stored procedures, user-defined functions or inline SQL statements) on the relational DataSource. This keeps mapping itself simple and unaffected by relational-specific things. Custom Tables provides an abstraction to mapping that allows commands in the database to be mapped to just as if they were physical tables, in most cases. Specifying this abstraction at the RSD level allows the user to deal with relational-specific concepts using relational terminology, rather than attempting to use domain-independent terminology at the mapping level.

**[0200]** For example, columns already exist in the relational domain. Custom Tables provides a CustomTable element where columns can be added just as in normal RSD Tables. This also helps keep mapping from becoming cluttered, and optimizes the solution by not introducing more places where the user must go to wire up the pieces.

**[0201]** Scenarios for Custom Tables include the following.

**[0202]** Adding a Condition to a Table/View. In this scenario, the user wants to add a filter to the base table. The canonical example is for the single table object-inheritance scenario where Person, Employee, Manager, etc., are all mapped to the same table and the user wants to filter the table based on the 'type' column. The user can specify a condition so that each type is mapped to its own table.

**[0203]** Add/Override Primary Key. Views by default do not have primary keys, but in order to use them effectively within the framework, a primary key field(s) must be defined on all structures. In order to achieve this, the user can create a simple Custom Table to specify the key field(s) on a view. The user can employ this same functionality to override the primary key on the underlying table or apply a virtual key to a procedure.

**[0204]** SingleComplexMapping. The user uses normal table mapping, but is faced with a limitation in mapping. To get around this, the user creates a one-off custom table. This scenario is distinct as the user only uses custom tables in limited areas and they should be able to use these in combination with normal physical table mapping.

**[0205]** Examples of this type of scenario include additional read-only columns on a UDF for Query, and additional write-only columns on an Update or Insert (userid, time updated, client-side calculated values, etc.).

**[0206]** ReadOnlyTables. The user has access to tables for read-only, but must use stored procedures for CUD. In this scenario, the user wants to base the Custom Table on an existing physical table, but override the Insert, Update and Delete operations to use stored procedures.

**[0207]** All Stored Procedures. The user cannot access tables at all and must use stored procedures and/or UDFs for all database access. In this scenario, there is no physical table on which to base the custom table.

**[0208]** Existing stored procedures. The scenarios above need to account for the situation where the user does not