Appendix

Aras.Server.Core.Configurator.IStringNotationConverter Interface Definition

The `Aras.Server.Core.Configurator.IStringNotationConverter` interface requires its implementors to support the `ConvertExpressionToStringNotation` method. This method provides a String Notation value to a given Effectivity Expression object representation.

The `Aras.Server.Core.Configurator.IStringNotationConverter` interface definition is:

```
namespace Aras.Server.Core.Configurator
{
        public interface IStringNotationConverter
        {
                string ConvertExpressionToStringNotation(ExpressionBase expression);
        }
}
```

Examples of Custom Effectivity String Notation

This section provides examples of customizing the Effectivity String Notation. These simple examples supplement the concepts detailed in sections *Customizing Effectivity String Notation on Effectivity Expression* and *Customizing Unified Effectivity String Notation*.

Custom Effectivity String Notation on Effectivity Expression

The default Effectivity String Notations Converter uses a conjunction word for each logical operator:

- **AND** for logical conjunction
- **OR** for logical disjunction
- **NOT** for logical complement

For example:

((Factory = Munich **AND** Production Date >= 01/01/2020) **OR**
(Factory = Detroit **AND** Production Date >= 06/01/2020)) **AND**
NOT (Body Type = Hatchback)

The customized String Notation will use the following characters instead of the conjunction words:

©2026 Aras Corporation All Copyrights Reserved

- **&&** for logical conjunction
- **||** for logical disjunction
- **!** for logical complement

For example:

((Factory = Munich **&&** Production Date >= 01/01/2020) **||**
(Factory = Detroit **&&** Production Date >= 06/01/2020)) **&&**
!(Body Type = Hatchback)

ExpressionToShortStringNotationConverter Class Definition

To achieve this customization goal, a new `ExpressionToShortStringNotationConverter` class is defined representing the new Converter:

©2026 Aras Corporation All Copyrights Reserved

```
internal class ExpressionToShortStringNotationConverter : Aras.Server.Core.Configurator.IStringNotationCo
{
    private readonly IEnumerable<Aras.Server.Core.Configurator.Variable> _variablesContainer;

    private static readonly Dictionary<Type, Aras.Server.Core.Configurator.DataType> typeTranslator =
        new Dictionary<Type, Aras.Server.Core.Configurator.DataType>
        {
            { typeof(string), Aras.Server.Core.Configurator.DataType.String },
            { typeof(int), Aras.Server.Core.Configurator.DataType.Int },
            { typeof(DateTime), Aras.Server.Core.Configurator.DataType.DateTime }
        };

    /// <summary>
    /// Initializes a new instance of the ExpressionToShortStringNotationConverter class with the defined
    /// </summary>
    /// <param name="variablesContainer">
    /// Represents a list of "Aras.Server.Core.Configurator.Variable" objects, which will be used to get
    /// human-readable identifiers of Variables and/or Named-constants.
    /// </param>
    public CustomExpressionToStringNotationConverter(IEnumerable<Aras.Server.Core.Configurator.Variable>
    {
        if (variablesContainer == null)
        {
            throw new ArgumentNullException("variablesContainer");
        }

        _variablesContainer = variablesContainer;
    }

    /// <summary>
    /// Converts an incoming "Aras.Server.Core.Configurator.ExpressionBase" Effectivity Expression object
    /// representation to a string notation and returns it.
    /// </summary>
    /// <param name="expression">An Effectivity Expression object representation.</param>
    /// <returns>A effectivity string notation value.</returns>
    public string ConvertExpressionToStringNotation(Aras.Server.Core.Configurator.ExpressionBase expressi
    {
        if (expression == null)
        {
            throw new ArgumentNullException("expression");
        }

        return TranslateToStringNotationImplementation(expression, null);
    }

    private string TranslateToStringNotationImplementation(
        Aras.Server.Core.Configurator.ExpressionBase expression,
        Aras.Server.Core.Configurator.ExpressionBase parentExpression)
    {
        if (expression.ExpressionType == Aras.Server.Core.Configurator.ExpressionType.Term)
        {
            return TranslateTermToStringNotation(expression);
        }

        string expressionStringNotation = string.Empty;
        string expressionOperatorStrNotation = GetAlgebraicNotationByOperator(expression.ExpressionType);
```

©2026 Aras Corporation All Copyrights Reserved

```
        // We assume at-least-one, at-most-one, and exactly-one expressions consist of terms
        // (<EQ><variable id="" /><named-constant id="" /></EQ>) only, here.
        if (expression.ExpressionType == Aras.Server.Core.Configurator.ExpressionType.AtLeastOne ||
            expression.ExpressionType == Aras.Server.Core.Configurator.ExpressionType.AtMostOne ||
            expression.ExpressionType == Aras.Server.Core.Configurator.ExpressionType.ExactlyOne)
        {
            IEnumerable<string> innerTermsStringNotation =
                expression.InnerExpressions.Select(expr => TranslateToStringNotationImplementation(expr,

            return string.Format(
                CultureInfo.InvariantCulture,
                expressionOperatorStrNotation,
                string.Join(" | ", innerTermsStringNotation));
        }

        foreach (Aras.Server.Core.Configurator.ExpressionBase innerExpression in expression.InnerExpressi
        {
            string innerExpressionStrNotation = TranslateToStringNotationImplementation(innerExpression,

            if (string.IsNullOrEmpty(innerExpressionStrNotation))
            {
                continue;
            }

            if (expression.ExpressionType == Aras.Server.Core.Configurator.ExpressionType.NOT)
            {
                expressionStringNotation += string.Format(
                    CultureInfo.InvariantCulture,
                    expressionOperatorStrNotation,
                    innerExpressionStrNotation);
            }
            else if (expression.ExpressionType == Aras.Server.Core.Configurator.ExpressionType.IMPLICATIO
            {
                string[] ifThen = expressionOperatorStrNotation.Split(' ');
                bool isStringNotationEmpty = string.IsNullOrEmpty(expressionStringNotation);

                // implicationTemplate describes "IF {expr}" or "THEN {expr}". In the "IF" case, a whites
                // after "{expr}" is required - "IF {expr} ".
                string implicationTemplate = isStringNotationEmpty ? "{0} {1} " : "{0} {1}";

                expressionStringNotation += string.Format(
                    CultureInfo.InvariantCulture,
                    implicationTemplate,
                    ifThen[isStringNotationEmpty ? 0 : 1],
                    innerExpressionStrNotation);
            }
            else
            {
                expressionStringNotation += !string.IsNullOrEmpty(expressionStringNotation)
                    ? expressionOperatorStrNotation + innerExpressionStrNotation
                    : innerExpressionStrNotation;
            }
        }

        bool shouldWrapInParentheses =
            parentExpression != null &&
```

©2026 Aras Corporation All Copyrights Reserved

```
            parentExpression.InnerExpressions.Count > 1 &&
            parentExpression.ExpressionType != Aras.Server.Core.Configurator.ExpressionType.IMPLICATION &
            expression.ExpressionType != Aras.Server.Core.Configurator.ExpressionType.NOT;

        return shouldWrapInParentheses ? "(" + expressionStringNotation + ")" : expressionStringNotation;
    }

    private string TranslateTermToStringNotation(Aras.Server.Core.Configurator.ExpressionBase expression)
    {
        var expressionTerm = expression as Aras.Server.Core.Configurator.ExpressionTerm;

        string variableId = expressionTerm.OperandLeft.GetId();
        Aras.Server.Core.Configurator.Variable variable =
            _variablesContainer.FirstOrDefault(var => var.Id == variableId);
        string variableName = variable?.Name ?? variableId;

        string assignedVariableValue = GetAssignedVariableValue(expressionTerm.OperandRight, variable);

        return string.Format(
            CultureInfo.InvariantCulture,
            "{0} {1} {2}",
            AddSquareBracketsIfNeed(variableName),
            GetAlgebraicTermOperatorNotation(expressionTerm.Operator),
            AddSquareBracketsIfNeed(assignedVariableValue));
    }

    private string GetAssignedVariableValue(
        Aras.Server.Core.Configurator.ITermOperand operandRight,
        Aras.Server.Core.Configurator.Variable variable)
    {
        if (operandRight.OperandType == Aras.Server.Core.Configurator.TermOperandType.NamedConstant)
        {
            string namedConstantId = operandRight.GetId();
            return variable?.Enum?.FindNamedConstantById(namedConstantId)?.Name ?? namedConstantId;
        }

        if (operandRight.OperandType == Aras.Server.Core.Configurator.TermOperandType.Constant)
        {
            object constantValue = operandRight.GetValue();
            Type constantType = constantValue.GetType();

            if (!typeTranslator.TryGetValue(constantType, out Aras.Server.Core.Configurator.DataType cons
            {
                throw new NotSupportedException(FormattableString.Invariant(
                    $"'{constantType.FullName}' constant data type is not supported as the right operand
            }

            // 'null' as the first parameter of the 'ToLocalString()' method call indicates the incoming
            // object shouldn't be adjusted to any time zone, because it's assumed 'datetime' is always r
            // in neutral format with UTC time zone.
            return constantDataType != Aras.Server.Core.Configurator.DataType.DateTime
                ? Convert.ToString(constantValue, CultureInfo.InvariantCulture)
                : Aras.I18NUtils.DateTimeConverter.ToLocalString(null, (DateTime)constantValue);
        }

        throw new NotSupportedException(FormattableString.Invariant(
            $"{operandRight.OperandType} is not supported as the right operand type for string notation c
```

©2026 Aras Corporation All Copyrights Reserved

```
    }

    private static string AddSquareBracketsIfNeed(string name)
    {
        return !string.IsNullOrEmpty(name) && !System.Text.RegularExpressions.Regex.IsMatch(name, @"^[a-z
            ? string.Format(CultureInfo.InvariantCulture, "[{0}]", name)
            : name;
    }

    private static string GetAlgebraicNotationByOperator(Aras.Server.Core.Configurator.ExpressionType boo
    {
        switch (boolOperator)
        {
            case Aras.Server.Core.Configurator.ExpressionType.AND:
                return " && ";
            case Aras.Server.Core.Configurator.ExpressionType.OR:
                return " || ";
            case Aras.Server.Core.Configurator.ExpressionType.NOT:
                return "!({0})";
            case Aras.Server.Core.Configurator.ExpressionType.IMPLICATION:
                return "IF THEN";
            case Aras.Server.Core.Configurator.ExpressionType.AtLeastOne:
                return "AT-LEAST-ONE({0})";
            case Aras.Server.Core.Configurator.ExpressionType.AtMostOne:
                return "AT-MOST-ONE({0})";
            case Aras.Server.Core.Configurator.ExpressionType.ExactlyOne:
                return "EXACTLY-ONE({0})";
            default:
                throw new NotSupportedException(boolOperator + " type is not supported for string notatio
        }
    }

    private static string GetAlgebraicTermOperatorNotation(Aras.Server.Core.Configurator.TermOperator ter
    {
        switch (termOperator)
        {
            case Aras.Server.Core.Configurator.TermOperator.Equal:
                return "=";
            case Aras.Server.Core.Configurator.TermOperator.GreaterThanOrEqual:
                return ">=";
            case Aras.Server.Core.Configurator.TermOperator.LessThanOrEqual:
                return "<=";
            default:
                throw new NotSupportedException(FormattableString.Invariant(
                    $"'{termOperator}' term operator is not supported for string notation conversion"));
        }
    }
}
```

Custom Unified Effectivity String Notation

By default, a comma (,) is used to combine multiple Effectivities on a single Relationship. A customized String Notation uses the OR conjunction word instead of the comma.

©2026 Aras Corporation All Copyrights Reserved

A customized String Notation example

Effective Item	Single Effectivity String Notations	Unified Effectivity String Notation	
		Default	Customized
Engine	Factory = Detroit	(Factory = Detroit), (Factory = Munich)	(Factory = Detroit) OR (Factory = Munich)
	Factory = Munich	(Factory = Detroit), (Factory = Munich)	(Factory = Detroit) OR (Factory = Munich)

JoinExpressionStringNotation Method Definition

To achieve this customization goal, change the definition of the `JoinExpressionStringNotation` internal method of the **effs_getExpressionStringNotation** Method as follows:

```
internal virtual string JoinExpressionStringNotation(IEnumerable<string> expressionStringNotations)
{
    if (!string.IsNullOrEmpty(expressionStringNotations.ElementAtOrDefault(1)))
    {
        expressionStringNotations = expressionStringNotations.Select(
            expressionStringNotation => "(" + expressionStringNotation + ")");
    }

    return string.Join(" OR ", expressionStringNotations);
}
```

Examples of Custom BOM Reports

This section provides examples of customizing BOM Reports. These simple examples supplement the concept described in section *Configuring BOM Reports*.

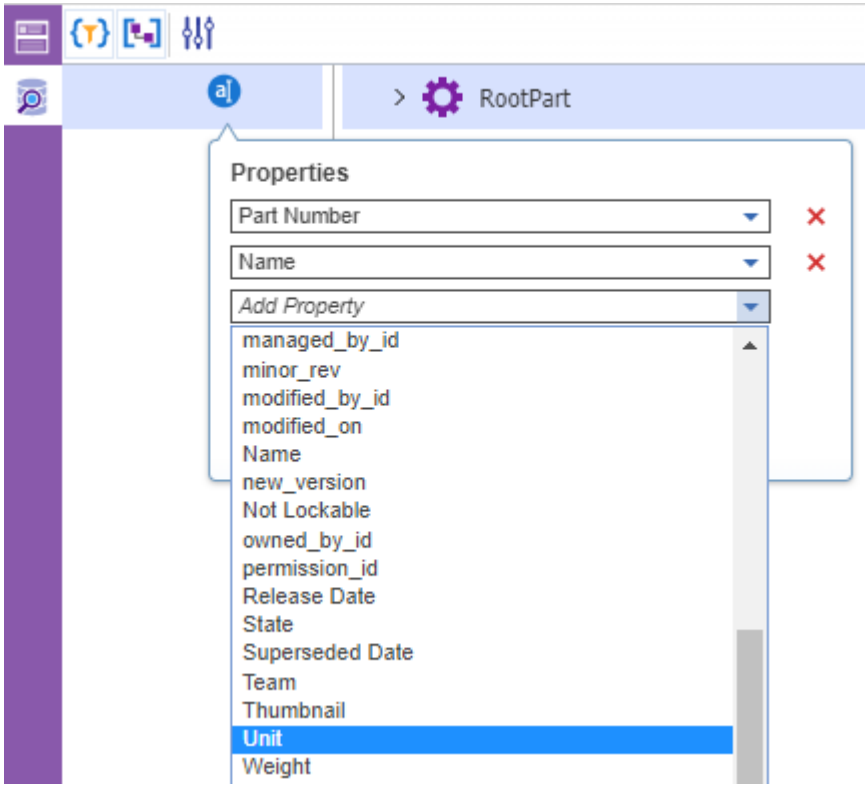
Adding a New Column Using an Item Property

In this example, the Multilevel BOM Report is customized. Using the following procedure, the **Unit** column is added between the **Quantity** and **AML Status** columns to display the **Unit** property of **Part** Items:

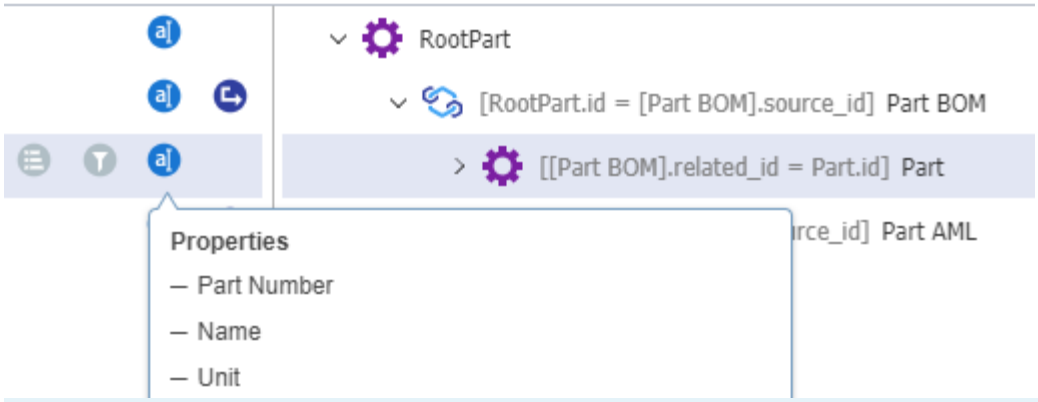
1. Search for the **PE_multilevelReport** Query Definition (QD) and open its item view for editing.
2. Click **Show Editor** on the **PE_multilevelReport** QD sidebar.
3. Click the **Properties** button on the top **RootPart** row. The **Properties** dialog box appears.

©2026 Aras Corporation All Copyrights Reserved

4. Add the **Unit** property to the **Properties** dialog box and click **Save**.



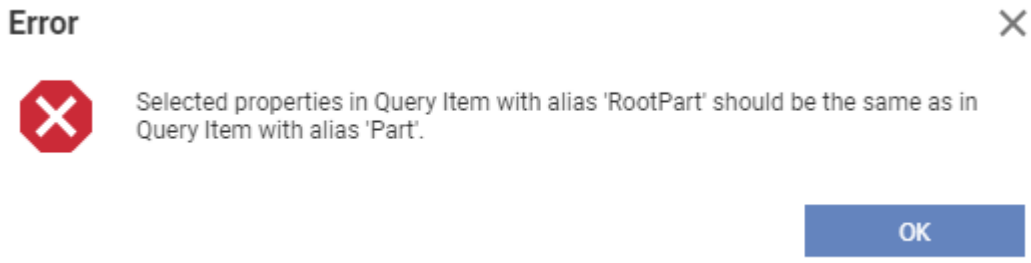
5. Repeat steps 3-4 on the **Part** row representing the Part Query Item.



Important

Both **Root** and **Part** Query Items should have the same Properties in a QD. Otherwise, Aras Innovator will return an error. This requirement does not apply to other Query Items of the QD.

©2026 Aras Corporation All Copyrights Reserved



6. Click **Save, Unlock & Close** on the **PE_multilevelReport** QD toolbar.
7. Search for the **Multilevel BOM Report** and open its Item view for editing.
8. Go to the **Stylesheet** editor tab.
9. Add the `<th>Unit</th>` table header cell element after `<th>Quantity</th>` .

```
<table border="0" cellspacing="0" cellpadding="0">
    <tr>
        <th colspan="{$MaxDepth + 1}">BOM Level</th>
        <th>Part Number</th>
        <th>Name</th>
        <th>Quantity</th>
        <th>Unit</th>
        <th>AML Status</th>
        <th>Manufacturer</th>
        <th>Manufacturer Part</th>
    </tr>
    <xsl:call-template name="rootItems"/>
</table>
```

10. Add a table data cell element for the `unit` property values after the one for `quantity` .

```
<td rowspan="{$rowCount}" width="120px">
<xsl:value-of select="unit"/>
</td>
```

```
<td rowspan="{$rowCount}" width="80px" align="center">
    <xsl:choose>
        <xsl:when test="depth=0">1</xsl:when>
        <xsl:when test="depth!=0 and (quantity)="">1</xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="quantity"/>
        </xsl:otherwise>
    </xsl:choose>
</td>
<td rowspan="{$rowCount}" width="120px">
    <xsl:value-of select="unit"/>
</td>
<td width="100px">
    <xsl:value-of select="Relationships/Item[@alias='Part AML'][1]/state"/>
</td>
```

©2026 Aras Corporation All Copyrights Reserved

11. Click **Apply** on the **Stylesheet** editor tab.
12. Click **Done** on the **Multilevel BOM Report** toolbar. The **Unit** column is added to the Multilevel BOM Report.

Bill of Materials Report

Generated on: 7/5/2019, 4:23:46 PM

BOM Level	Part Number	Name	Quantity	Unit	AML Status	Manufacturer	Manufacturer Part
0	Parent Part	Parent Part	1	EA			
1	Child Part A	Child Part A	1	EA			
2	Child Part A.1	Child Part A.1	1	EA			
2	Child Part A.2	Child Part A.2	1	EA			
1	Child Part B	Child Part B	1	EA			
2	Child Part B.1	Child Part B.1	1	EA			
2	Child Part B.2	Child Part B.2	1	EA			

Adding a New Column Using a Relationship Property

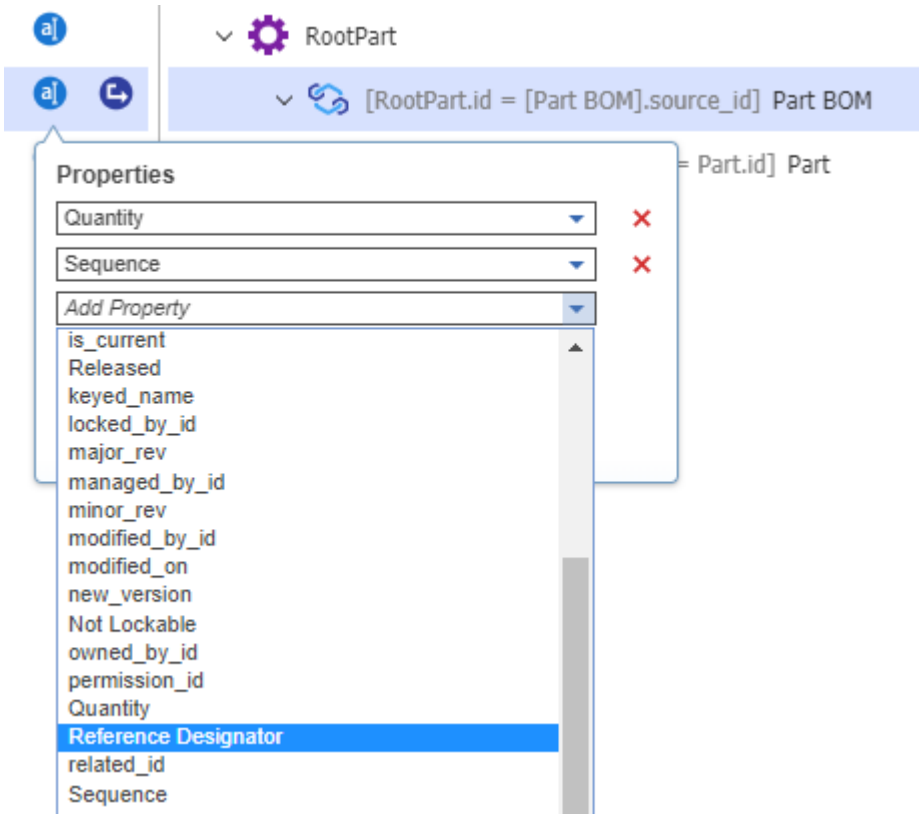
Important

This example can be used as a guide for adding any relationship property. For the **Sequence** relationship property, prefer the configuration as described in the next section.

In this example, the BOM Costing Report is customized. Using the following procedure, the **Reference Designator** column is added as the last column to display the **Reference Designator** property of the **Part BOM** Relationship ItemType:

1. Search for the **PE_costingReport** Query Definition (QD) and open its item view for editing.
2. Click **Show Editor** on the **PE_costingReport** QD sidebar.
3. Click the **Properties** button on the **Part BOM** row. The **Properties** dialog box appears.
4. Add the **Reference Designator** property to the **Properties** dialog box and click **Save**.

©2026 Aras Corporation All Copyrights Reserved



5. Click **Save, Unlock & Close** on the **PE_costingReport** QD toolbar.
6. Search for the **BOM Costing Report** and open its Item view for editing.
7. Go to the **Stylesheet** editor tab.
8. Add the `<th>Reference Designator</th>` table header cell element after `<th>Cost Basis</th>` .

```
<div class="table_wrapper">
    <div class="report_name">
        Bill of Materials Costing Report
    </div>
    <div class="report_generation_time"/>
    <table border="0" cellspacing="0" cellpadding="0">
        <tr>
            <th colspan="{$MaxDepth + 1}">BOM Level</th>
            <th>Part Number</th>
            <th>Name</th>
            <th>Quantity</th>
            <th>Cost</th>
            <th>Cost Basis</th>
            <th>Reference Designator</th>
        </tr>
        <xsl:call-template name="Levels"/>
    </table>
</div>
```

©2026 Aras Corporation All Copyrights Reserved

9. Add a table data cell element for the `reference_designator` property values after the one for `cost_basis` .

```
<td>
<xsl:value-of select="Relationships/Item[@alias='Part BOM' and @info='parent']/reference_designator"/>
</td>
```

Important

When customizing the Multilevel BOM Report with a Relationship ItemType property, this property table data tag should have the rowspan attribute as follows: *<td rowspan="{$rowCount}" width="120px" > <xsl:value-of select="Relationships/Item[@alias='Part BOM' and @info='parent']/property_name"/ > </td >*

```
                <td>
                    <xsl:value-of select="cost_basis"/>
                    <xsl:if test="cost_basis='' or not(cost_basis)">
                      <xsl:text> </xsl:text>
                    </xsl:if>
                </td>
                <td>
                    <xsl:value-of select="Relationships/Item[@alias='Part BOM' and @info='parent']/reference_designator"/>
                </td>
            </tr>
        </xsl:template>
</xsl:stylesheet>
```

10. Click **Apply** on the **Stylesheet** editor tab.
11. Click **Done** on the **BOM Costing Report** toolbar. The **Reference Designator** column is added to the BOM Costing Report.

Bill of Materials Costing Report

Generated on: 10/15/2019, 12:50:33 PM

BOM Level	Part Number	Name	Quantity	Cost	Cost Basis	Reference Designator
0	Parent Part	Parent Part	1	16.0000	Calculated	
1	Child Part A	Child Part A	1	10.0000	Calculated	RD P-A
2	Child Part A.1	Child Part A.1	1	7.0000	Actual	RD A-A1
2	Child Part A.2	Child Part A.2	1	3.0000	Actual	RD A-A2
1	Child Part B	Child Part B	1	6.0000	Calculated	RD P-B
2	Child Part B.1	Child Part B.1	1	2.0000	Actual	RD B-B1
2	Child Part B.2	Child Part B.2	1	4.0000	Actual	RD B-B2

Adding the Sequence Column

©2026 Aras Corporation All Copyrights Reserved

In this example, the BOM Costing Report is customized. Using the following procedure, the **Sequence** column is added as the last column to show the **Sequence** property of the **Part BOM** Relationship ItemType:

1. Search for the **BOM Costing Report** and open its Item view for editing.
2. Go to the **Stylesheet** editor tab.
3. Add the `<th>Sequence</th>` table header cell element after `<th>Cost Basis</th>` .

```
<div class="table_wrapper">
    <div class="report_name">
        Bill of Materials Costing Report
    </div>
    <div class="report_generation_time"/>
    <table border="0" cellspacing="0" cellpadding="0">
        <tr>
            <th colspan="{$MaxDepth + 1}">BOM Level</th>
            <th>Part Number</th>
            <th>Name</th>
            <th>Quantity</th>
            <th>Cost</th>
            <th>Cost Basis</th>
            <th>Sequence</th>
        </tr>
        <xsl:call-template name="Levels"/>
    </table>
</div>
```

4. Add a table data cell element for the `sort_order` property values after the one for `cost_basis` .

```
<td>
<xsl:value-of select="sort_order"/>
</td>
```

Important

When customizing the Multilevel BOM Report with the Sequence property, this property table data tag should have the rowspan attribute as follows: *<td rowspan="{$rowCount}" width="120px" > <xsl:value-of select="sort_oder"/ > </td >*

©2026 Aras Corporation All Copyrights Reserved

```
                <td>
                    <xsl:value-of select="cost_basis"/>
                    <xsl:if test="cost_basis='' or not(cost_basis)">
                      <xsl:text> </xsl:text>
                    </xsl:if>
                </td>
                <td>
                    <xsl:value-of select="sort_order"/>
                </td>
            </tr>
        </xsl:template>
</xsl:stylesheet>
```

5. Click **Apply** on the **Stylesheet** editor tab.
6. Click **Done** on the **BOM Costing Report** toolbar. The **Sequence** column is added to the BOM Costing Report.

Bill of Materials Costing Report

Generated on: 10/15/2019, 1:33:40 PM

BOM Level	Part Number	Name	Quantity	Cost	Cost Basis	Sequence
0	Parent Part	Parent Part	1	16.0000	Calculated	
1	Child Part A	Child Part A	1	10.0000	Calculated	1
2	Child Part A.1	Child Part A.1	1	7.0000	Actual	1
2	Child Part A.2	Child Part A.2	1	3.0000	Actual	2
1	Child Part B	Child Part B	1	6.0000	Calculated	2
2	Child Part B.1	Child Part B.1	1	2.0000	Actual	1
2	Child Part B.2	Child Part B.2	1	4.0000	Actual	2

©2026 Aras Corporation All Copyrights Reserved