List of Figures

Figure 5–1: Solving Problems with Routing Rules (Drawing A) .......................................................... 32
Figure 5–2: Solving Problems with Routing Rules (Drawing B) .......................................................... 33
Figure 5–3: Solving Problems with Routing Rules (Drawing C) .......................................................... 34
Figure 5–4: Solving Problems with Routing Rules (Drawing D) .......................................................... 35
About This Book

This book explains how to define business rules that direct business process logic. It contains the following chapters:

• About Business Rules
• Introduction to the Rule Editor
• Creating and Editing Rule Sets
• Example Rule Classes
• Debugging Routing Rules
• Ensemble Utility Functions

For a detailed outline, see the table of contents.

The following books provide related information:

• Ensemble Best Practices describes best practices for organizing and developing Ensemble productions.
• Developing Ensemble Productions explains how to perform the development tasks related to creating an Ensemble production.
• Configuring Ensemble Productions explains how to perform the configuration tasks related to creating an Ensemble production.
• Monitoring Ensemble describes how to monitor Ensemble. In particular, see the chapter “Viewing the Business Rule Log.”

If you are using an Ensemble production to route and transform messages in an Electronic Data Interchange (EDI) format such as HL7 or X12, see the books in the Ensemble Interoperability set.

For general information, see the InterSystems Documentation Guide.
Business rules allow nontechnical users to change the behavior of Ensemble business processes at specific decision points. You can change the logic of the rule instantly, using the Ensemble Rule Editor in the Management Portal. There is no need for programming or diagramming skills to change the rule, and there is no need to modify or compile production code for changes to take effect. The following figure shows how business rules work.

Suppose an enterprise runs an Ensemble production that processes loan applications consistently across an international enterprise. The decision process is consistent worldwide, but banks must adjust the local acceptance criteria from country to country. Business rules support this division of responsibility as follows:
1. The developer of the business process identifies a decision point, by naming the business rule that will make the decision on behalf of the business process. The developer leaves a placeholder for that business rule in the Business Process Language (BPL) code by invoking the Business Process Language (BPL) element `<rule>`. The `<rule>` element specifies the business rule name, plus parameters to hold the result of the decision and (optionally) the reason for that result. Suppose we call this rule LoanDecision.

2. Wherever the `<rule>` element appears in a BPL business process, a corresponding rule definition must exist within the production. A user at the enterprise, typically a business analyst, may define the rule using a browser-based online form called the Ensemble Rule Editor. This form prompts the user for the simple information required to define the business rule called LoanDecision. Ensemble saves this information in its configuration database.

   Any enterprise user who is familiar with the Ensemble Rule Editor and who has access to it in the Management Portal can modify the rule definition. Modifications are simply updates to the database and can be instantly applied to a production while it is running. Therefore, it is possible for business analysts at various regional locations to run the Ensemble Rule Editor to modify their copies of the rule to provide different specific criteria appropriate to their locales.

3. At runtime, upon reaching the BPL `<rule>` statement the business process invokes the rule named LoanDecision. The rule retrieves its decision criteria from the configuration database, which may be different at different locales. Based on these criteria, the rule returns an answer to the business process. The business process redirects its execution path based on this answer.

4. For ongoing maintenance purposes, the business process developer need not be involved if a rule needs to change. Any rule definition is entirely separate from business process code. Rule definitions are stored in the Ensemble configuration database as classes and are evaluated at runtime. Additionally, rule definitions can be exported and imported from one Ensemble installation to another.

   In this way, enterprise users such as business analysts can change the operation of the business process at the decision point, without needing the programming expertise that would be required to revise the BPL or class code for the business process.

Ensemble provides four types of business rule definition:

- **General Business Rule**
- **HL7 Message Routing Rule**
- **General Message Routing Rule**
- **Virtual Document Message Routing Rule**

Each type has an associated rule assist class that controls how the editor works and provides a Rules Assistant in the right pane while you are editing.

You create rules for a definition that are grouped into rule sets. Each rule definition has one or more rule sets. Each rule set is associated with a beginning and ending effective date and time. Each time a process invokes a rule, one and only one rule set is executed.

Note that there is overlap among the options available in business processes, data transformations, and business rules. For a comparison, see “Comparison of Business Logic Tools” in Developing Ensemble Productions.

The remaining chapters in this book describe how to define business rules including how to create and use rule sets using the Ensemble Rule Editor as well as how to invoke rules using BPL and using business process routing engines.
1.1 Rules as Classes

The Ensemble Rule Editor provides a structured way for enterprise business analysts to shape the logical decisions made by a business process, without needing any programming skills.

However, a business process developer can instead work with business rule definitions as classes, using Studio. The following figure shows this format.

```xml
/// Business rule responsible for mapping an input location
/// to "F" for Fahrenheit or "C" for Celsius temperature scale.
{
  Parameter RuleAssistClass = "Ens.Rule.GeneralBusinessRuleAssist";

  XData RuleDefinition [ XMLNamespace = "http://www.intersystems.com/rule" ]
  {
      <ruleSet name="" effectiveBegin="" effectiveEnd="">
        <rule name="" disabled="false">
          <when condition="(Location=&quot;US&quot;)||/(Location=&quot;USA&quot;)">
            <return>"F"</return>
          </when>
        </otherwise>
        <return>"C"</return>
      </otherwise>
    </rule>
  }
}
```

You can open a business rule as a class in Studio, edit the document, and save the changes. Changes saved in Studio are immediately visible in the Ensemble Rule Editor. If you do not see your changes, simply refresh the page.

**Package Mapping Rule Classes**

Since they are classes, you can map rules to other namespaces. If you do map rule classes, you must recompile all the mapped rule classes in each namespace where you use them to insure the local metadata is available in each namespace.

If you are using rules that have been upgraded from legacy rules, you may encounter compile errors due to rule aliases not being unique. You must fix these compiler errors manually.

For details, see “Package Mapping” in the chapter “Packages” in *Using Caché Objects*. 

2

Introduction to the Rule Editor

This chapter introduces the Rule Editor in the Management Portal. It is divided into the following sections:

- Business Rule List
- Business Rule Editor

2.1 Business Rule List

The Business Rule List page displays a list of the business rule classes defined in the active Ensemble namespace. Navigate to this page from the Business Rules item of the Ensemble List menu.

Select a rule class to be the target of one of the following commands in the ribbon bar:

- **Edit** — Click to change or view the rule definition using the Business Rule Editor.
- **Delete** — Click to permanently delete the rule definition class.
- **Export** — Click to export the selected rule class as an XML file.
- **Import** — Click to import an XML file into a rule class.

You can also export and import rule classes as you do any other class in Ensemble. You can use the Globals page of the Management Portal (System Explorer > Globals) or use the Export and Import commands on the Tools menu in Studio.

2.2 Business Rule Editor

The Ensemble Rule Editor page is where you create and edit business rule class definitions for all types of business rule. The page opens with the last rule you had open in the namespace. The tab at the left of the title bar contains the name of the business rule definition class. If this is the first time on the page for this namespace, the working pane is empty and you must either create a new rule or open an existing one.

The “Creating and Editing Rule Sets” chapter describes the details of how to use the editor to define business and routing rules; the rest of this chapter describes how the user interface works with the Ensemble rule structure.

The ribbon bar of the Ensemble Rule Editor page contains the following elements:

- **New** button — Click to launch the Business Rule Wizard to create a new business rule definition.
• **Open** button — Click to launch the Finder Dialog to choose an existing business rule definition to edit.

• **Save** button — Click to save and compile any changes you have made to the rule definition.

• **Save As** button — Click if you have been editing a rule definition and wish to save your changes as a new business rule class.

• **Contract** — Click to contract the display of all of the rules in the rule set. You can then individually expand the rule you want to view or edit.

• **Expand** — Click to expand the display of all of the rules in the rule set.

• Open new windows checkbox — If checked, the New and Open commands open the rule editor in a new window or browser tab.

• Zoom drop-down—Specifies the zoom percentage to view the rule.

**General Tab**

Once you have a rule definition in the working pane you see tabs of information. The general tab contains the summary information for the rule definition:

**Description**

Class description of the rule definition and its purpose.

**Rule Type / Rule Assist Class**

Each rule type has an associated rule assist class which controls the constraints of the rule and provides information in the right pane of the page to guide you when editing rules.

The table shows the four rule types and their associated Ens.Rule.Assist class:

<table>
<thead>
<tr>
<th>Rule Type</th>
<th>Rule Assist Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Message Routing Rule</td>
<td>EnsLib.MsgRouter.RuleAssist</td>
</tr>
</tbody>
</table>

**Context Class**

The class that contains the information to tell the Ensemble Rule Editor which object properties to provide as choices in the Value editor while you are editing a rule. For general rules, it is generated from the business process BPL class and ends in .Context. For routing rules without a BPL process, it is usually the routing engine business process class.

**Rule Alias**

This field may have a value if a rule was created in older releases before Ensemble stored rule definitions as classes. When the rule is upgraded, if the old name does not conform to class naming conventions, it becomes the alias. You can use this field for continuing such naming conventions, but if you are creating a new rule you do not need to use this field.

**Temporary Variables**

You can specify temporary variables in this field. You can use these temporary variables in the business rule. Each variable specification is separated by commas. For example:
Temporary variables are used in a rule by preceding the variable name with an @ (at sign). For example, @FreeShippingValue. Temporary variables are available only within the rule. If you want to pass information to the transformation using the rule, use the RuleUserData property. See “Selecting the Transformation and Target of a Send Action” for details.

Production Name
(Only for routing rules) This optional, informational setting makes it easier for you to define rules because the rule editor uses this setting to select configuration items to display when you are defining rules. The production configuration actually specifies that a rule is used for a production.

Rule Set List
List of Rule Sets with the following information:

- Rule Set Name — Name to identify the particular set.
- Beginning Date and Time — The time from which the rule becomes active. The exact time is included in the active interval. The format is YYYY-MM-DDTHH:MM:SS. The time portion is optional and defaults to 00:00:00.
- Ending Date and Time — The time when the rule stops being active. The exact time is excluded from the active interval. The format is YYYY-MM-DDTHH:MM:SS. The time portion is optional and defaults to 24:00:00.

Rule Set Tabs
Each rule set has its own tab for editing its list of rules. For details, see the “Creating and Editing Rule Sets” chapter.

Editor Icons
Both tabs contain following set of icons. The following icons are available to edit the rule definition, rule sets, rules, and clauses within a rule set:

- Click the Up icon to move the selected item up in the list.
- Click the Down icon to move the selected item down in the list.
- Click the Add icon to add the type of item you choose from the list or beneath the icon in the Rule Assistant.
- Click the Delete icon to delete the item next to it.
- Click the Undo icon to undo the last change.
- Click the Redo icon to redo the last change that was undone.
Click the Function icon to open the associated editor for the selected action.

If an action is not available, its icon appears dimmed.

**Rule Assistant**

You can hide or show the Rule Assistant using the double-arrow in the right pane of the Ensemble Rule Editor. When you are editing a rule set, the expanded **Rule Assistant** pane provides you with help throughout the editing process. It describes the item you have selected and provides a list of options based on your assist class.

### 2.2.1 Business Rule Wizard

This wizard helps you create a new business rule definition based on the **Ens.Rule.Definition** class with an XData block named **RuleDefinition**. Enter values for the following fields:

**Package**

Enter a package name or use the arrow to select an existing package name.

**Name**

Enter the name of the business rule class.

**Alias**

*(Optional)* Enter the alias name for this rule. Do not use any of the following characters:

; , : ! * - $ ' " < > 

Generally, this exists for some rules created in older releases before Ensemble stored rule definitions as classes and for continuing such naming conventions. If you are creating a new rule you do not need to use this field.

**Description**

*(Optional)* Enter a description for this rule definition. This becomes the class description.

**Type**

Enter one of four rule types:

- General Business Rule
- HL7 Message Routing Rule
- General Message Routing Rule
- Virtual Document Message Routing Rule

Each rule type has an associated rule assist class which provides information in the right pane of the page to guide you in entering rules and controls which options the editor presents.

**Context Class**

*(Optional)*
This field tells the Ensemble Rule Editor which object properties to provide as choices in the Value field when you are editing a rule. For general rules, this class is generated from the BPL business process class that invokes the `<rule>` tag. The naming convention of the class is the business process class name plus the `.Context` extension, as in Demo.ZenService.Bproc.WeatherReport.Context.
3
Creating and Editing Rule Sets

This chapter describes how to use the Ensemble Rule Editor to develop rule sets.

There are two types of rule set:

- **General business rule set** — A list of rules that are evaluated sequentially until one of them is found to be true. The “true” case determines the next action of the business process that invoked the rule. If none of the rules is true, the rule set returns a default value. This is the type of rule that you invoke using the BPL <rule> element.

- **Routing rule set** — A rule set for use in message routing productions. Based on the type and contents of incoming messages (constraint), the routing rule set determines the correct destination for each message and how to transform the message contents prior to transmission. You use a routing engine business process to invoke this rule set.

**Rule Set Properties**

When you add a rule set, you see three property fields that you can use to distinguish rule sets in a rule definition: Name, Effective begin date and time, and Effective end date and time. These are the values the rule set list shows on the general tab.

Most business rule definitions have only one rule set that is always in effect. You can, however, have more than one version of a rule defined and become active at different times using a beginning and ending effective date and time. Each time a process invokes a rule, one and only one rule set is executed.

You can add a rule set from the general tab by clicking the add icon with the rule set list selected. You can then begin to edit the rule set by clicking on its tab or double-clicking its row in the list. When you are editing a rule set, you can click the icons near the top of the rule set tab or you can click the ones (or in the case of the add icon, the labeled rectangles beneath) in the expanded Rule Assistant pane.

As you become more familiar with the rule set editor and the rule sets you are developing, you may find it unnecessary to view the property names throughout the display of the rule set. You can toggle the viewing of property names by clicking the green square in the top right of the editor pane.

Also throughout the editing process, if a property does not contain a valid value you see a small red circle containing an exclamation point at the top right of the property box. If you double-click this warning mark, a helpful error message displays.

The following sections describe the editing tasks involved in creating a rule set:

- Adding Rules
- Adding Actions
- Using the Associated Editors
- Adding Business Rule Notification
3.1 Adding Rules

Each rule set contains one or more rules that you define to satisfy a specific function in a business process.

**Rule Properties**

When you add a rule, you see the following properties:

**Name**

You can give this rule an optional name to help you identify it.

Internally, Ensemble names the rules in sequential order in the form rule#n. If you enter a value in the Name property, it appears in the class definition and also appears in parentheses next to the internal rule name in the rule log. The value of n changes if you reorder the rules in a rule set.

**Disabled**

You can double-click this item to toggle between disabling and enabling the rule. A value of true means the rule is disabled and therefore skipped when the rule set is executed.

**Constraint**

*(For routing rules only)* — The constraint property distinguishes a routing rule. As a message makes its way through the rule set, if it matches the constraint you define for the rule, that rule logic is executed. See “Using the Rule Constraint Editor” for details on defining the constraint.

Each rule consists of a series of one or more when clauses and an optional otherwise clause, along with some optional actions. When you add a rule, the editor starts you with a when clause and if it is a routing rule, it also provides a return action for the clause.

Some general considerations to keep in mind when developing rules in your rule set:

- Once the execution through a rule set encounters a return action, the execution of the rule set ends and returns to the business process that invoked the rule definition class.

- You can control the execution of more than one rule in a rule set by omitting the returns. In other words, if you want to check all rules, do not provide a return action within any of the rule clauses. You may then provide a value in a return action at the end of the rule set for the case where no rule clauses evaluate to true.

- When a rule contains multiple when clauses, only the actions indicated by the first when that evaluates to true are performed. You can use an otherwise clause to perform an action if no when conditions are true.

- Each when clause has a condition property. A common design for a general business rule set is one that contains one rule with a series of when conditions and returning a value depending on which condition is true. If you want to return a default value if none of the conditions is true, you can use the otherwise clause with a return.

- A common design for a routing rule set is one that contains several rules each with a different constraint defined and each with one when clause describing how and where to route the message that matches the constraint.

3.2 Adding Actions

Every clause within a rule can have zero or more actions associated with it. Actions are executed if and only if the associated when condition is true. You can add the following actions to a rule set or a when or otherwise clause within a rule:
### Assigns values to properties in the business process execution context.

Assigns values to properties in the business process execution context. For details see the `<assign>` entry in the *Ensemble Business Process Language Reference*.

### Returns to the business process without further execution of the rule.

Returns to the business process without further execution of the rule. For general rules it also returns the indicated value to the result location.

### Adds the information you enter into the Event Log when this specific part of the rule is executed.

Adds the information you enter into the Event Log when this specific part of the rule is executed. For details see the `<trace>` entry in the *Ensemble Business Process Language Reference*.

### Adds the expression text and value to the Rule Log when this specific part of the rule is executed.

The `debug` action is executed only if the router business process RuleLogging property specifies the `d` flag. For details on the RuleLogging property, see “Rule Logging” in *Ensemble Virtual Documents*

You can add some actions at the rule set level, but they do not always logically make sense. You should contain most actions within the `when` clauses of rules. A time when it may make sense to provide an action outside of these clauses is to set a default return value if no rules are executed in a rule set.

### 3.3 Using the Associated Editors

When you select a property of any of the items in a rule definition, $f_x$ (the function icon) becomes enabled if the property has an associated editor. The following table shows which properties exist for a item, clause, or action and which editor opens when you double-click the property or click $f_x$.

<table>
<thead>
<tr>
<th>Item</th>
<th>Property</th>
<th>Associated editor or edit action</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule set</td>
<td>name</td>
<td>Enter text.</td>
</tr>
<tr>
<td></td>
<td>effective begin</td>
<td>Date and Time Selector</td>
</tr>
<tr>
<td></td>
<td>effective end</td>
<td></td>
</tr>
<tr>
<td>rule</td>
<td>name</td>
<td>Enter text.</td>
</tr>
<tr>
<td></td>
<td>disabled</td>
<td>Double-click to toggle between true and false.</td>
</tr>
<tr>
<td></td>
<td>constraint</td>
<td>Rule Constraint Editor</td>
</tr>
<tr>
<td>when</td>
<td>condition</td>
<td>Expression Editor</td>
</tr>
</tbody>
</table>

You can add the following actions to a routing rule:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>send</td>
<td>Sends the message to a particular target after optionally transforming it.</td>
</tr>
<tr>
<td>delete</td>
<td>Deletes the current message.</td>
</tr>
<tr>
<td>delegate</td>
<td>Delegates the message to a different rule.</td>
</tr>
</tbody>
</table>

In addition, you can add the following actions to a routing rule:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>send</td>
<td>Sends the message to a particular target after optionally transforming it.</td>
</tr>
<tr>
<td>delete</td>
<td>Deletes the current message.</td>
</tr>
<tr>
<td>delegate</td>
<td>Delegates the message to a different rule.</td>
</tr>
</tbody>
</table>
Creating and Editing Rule Sets

### Item | Property | Associated editor or edit action
--- | --- | ---
assign | property | Enter the name of a context property that is the target of this assignment. This must be a property in an execution context object.
 | value | Expression Editor
return | value (general rule set only) | Expression Editor
trace | value | Expression Editor
send | transform | Data Transform Selector
 | target | Production Configuration Item Selector
delegate | rule name | Finder Dialog for rule classes.

The **otherwise** clause and **delete** action have no properties to edit.

### 3.3.1 Using the Rule Constraint Editor

Routing rules have a constraint property you use to determine which messages to route through which rules. Use this editor to configure the rule constraint values, which can be made up of the following properties:

**Source**

Configuration name of one of the following items:

- A business service (for a routing interface)
- A message routing process (if another rule chains to this routing rule set)

Click (…) next to the **Source** field to invoke the **Production Configuration Item Selector** which displays a list of possible source items in the production you indicate on the general tab. If you have not yet prepared the item you need as a **Source**, you may leave this field blank and return to it when the item is ready.

**Message Class**

Identifies the Ensemble message object that is being routed by this rule. The value of this field depends on the routing rule type:

- **HL7 Message Routing Rule** — Defaults to EnsLib.HL7.Message; you do not have the option of entering this property in a new rule definition.
- **General Message Routing Rule** — Click … next to the **Message Class** field to invoke the **Finder Dialog** to select the appropriate message class. You can choose the category of message class to narrow your choices.
- **Virtual Document Message Routing Rule** — Choose from the list of defined virtual document classes.

The following fields in the editor apply only when the you are editing an HL7 or virtual document routing rule class, such as X12 or ASTM. For general message routing rules, you are finished entering the constraint fields.

**Schema Category**

Identifies the message category for the particular message class:

- **HL7 Message Routing Rule** — Choose from the built-in schema category list or the name of a custom schema definition.
• **Virtual Document Message Routing Rule** — Choose from the list of category types defined for your chosen virtual document class; they could be built-in or from an imported custom schema.

**Document Name**

Identifies the message structure; the acceptable values depend on the message class

- **HL7 Message Routing Rule** — The HL7 message structure that the source application identifies in the MSH:9 field, such as ADT_A08 or ORM_O01. To allow easy retrieval, this MSH:9 value resides in the EnsLib.HL7.Message property called Name.

- **Virtual Document Message Routing Rule** — Choose from the list of category types defined for your chosen virtual document class; they could be built-in or imported.

Enter more than one value in the **Document Name** text entry field. This causes the rule to match any of the specified **Document Name** values, and no others.

If you leave any of the fields blank, Ensemble considers all values to be a match for that rule.

The **Constraint Editor** behaves somewhat differently when you are editing a rule set converted from a version older than Ensemble 2012.1. You may see the **Schema Doc Type** field with a box to select an item to append to the list.

**Schema Category** and **Schema Doc Type** represent the actual HL7 message structure. These values reside in the EnsLib.HL7.Message property called DocType. DocType is a two-part string separated by a colon, such as 2.4:ADT_A08.

- At left is the **Schema Category**. This is the name of a built-in schema category or the name of a custom schema definition.
- At right is the **Schema Doc Type**. This is an HL7 message structure within the identified schema, such as ADT_A08 or ORM_O01.

### 3.3.2 Using the Production Configuration Item Selector

This editor helps you choose a configuration item as a source of a message or a routing target of a message. You choose from a list of production configuration items defined in the production you enter in the general tab. If you do not choose a production in the general tab of your rule definition, you receive the following warning when you invoke this editor:

No production name has been specified in the General tab of your rule. Be careful and ensure that your chosen target(s) exist in your production.

In this case you choose from a list of production configuration items defined in the current namespace. While developing your production rules, be careful to verify the names of your configuration items.

### 3.3.3 Selecting the Transformation and Target of a Send Action

When you add a **send** action, you also enter the following properties:

- **Transform** — *(optional)* To transform the message before sending it to the **Target**, enter the full package and class name of one or more DTL data transformations. You can double-click the **Transform** field to invoke the **Data Transform Selector** to choose one or more defined data transformations in the namespace.

  Multiple data transformations are chained in the order in which they appear, from left to right.

- **Target** — Enter the configured name of one or more of the following production items:
  - A business operation, to route the message to an external application
  - A routing process, to chain to another routing rule set
Double-click the **Target** field to invoke the **Production Configuration Item Selector** to choose one or more production configuration items.

If you enter items for these field that do not exist yet, make sure to verify you have entered the correct name when the production does contain them.

If you want to pass information to the rule, you can assign a value to the `RuleUserData` property. This value is accessible to the transformation in the `aux.RuleUserData`. For details on using the `aux` variable, see “Valid Expressions” in the chapter “Syntax Rules” in *Developing DTL Transformations*.

### 3.3.4 Using the Expression Editor

When you select a condition or a value and click $f_x$, you invoke the Expression Editor. There are four properties that activate the expression editor:

- **when condition** — See *Defining When Conditions* for details.
- **assign value**
- **return value of a general business rule**
- **trace value** — This is the text for the trace message. It can be a literal text string or an expression to be evaluated. If an expression, it must use the scripting language specified by the `language` attribute of the containing `<process>` element.

When defining an expression, you can nest several conditions by using the icons described in the following table.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>Click to move the selected node up in the expression.</td>
</tr>
<tr>
<td>↓</td>
<td>Click to move the selected node down in the expression.</td>
</tr>
<tr>
<td>←</td>
<td>Click to merge the selected node with the parent node.</td>
</tr>
<tr>
<td>$op$</td>
<td>Click to choose from a list of operators of which to make the selected node an operand.</td>
</tr>
<tr>
<td>$f_x$</td>
<td>Click to choose from a list of Ensemble functions to make the selected node an argument of the selected function.</td>
</tr>
<tr>
<td>+</td>
<td>Click to add a sibling node.</td>
</tr>
<tr>
<td>✗</td>
<td>Click to delete the selected node.</td>
</tr>
</tbody>
</table>

If an action is not available, its icon appears dimmed. As you add conditions and values to the expression diagram, you see the text of the expression in the blue bar at the top of the editor.

The following sections provide greater detail for entering expression values:

- **Expression Values**
- **Expression Operators**
- **Expression Functions**
3.3.4.1 Defining When Conditions

Within a rule definition, a condition consists of two values and a comparison operator between these values. For example:

\[ \text{Amount} \leq 5000 \]

If a condition is not true, it is false. There are no other possible values for a condition. This type of result is called a Boolean result. Ensemble stores a Boolean result either as the integer value 1 (if true) or 0 (if false). In most cases you do not need to be concerned with this internal representation; however, when using the constraint property in a routing rule, you may want to always execute the associated \textit{when} clause when the constraints are satisfied. In this case, enter a value of 1 in the \textit{when} condition property.

There can be more than one condition within a rule. If so, all of the conditions must be evaluated and compared before the rule (as a whole) can be found to be true or false. The logic between each condition is controlled using AND or OR operators. For example:

\[
\begin{align*}
\text{IF } & \text{ Amount} \leq 5000 \\
& \text{ AND CreditRating} > 5 \\
& \text{ OR CurrentCustomer} = 1
\end{align*}
\]

For this example, the \textbf{Expression Editor} dialog appears as follows:

The preceding rule has three conditions: Amount \( \leq 5000 \), CreditRating \( > 5 \), CurrentCustomer = 1. Each of these conditions could be true or false. All of these conditions are evaluated before the AND or OR operators come into play.

AND and OR operate on true and false values only. The operator is positioned between two Boolean values, and returns a single Boolean result based on those two values, as follows.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Result is true when...</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Both values are true.</td>
</tr>
<tr>
<td>OR</td>
<td>At least one of the values is true, or both are true. If one of the values is false and the other is true, and the result (as a whole) is still true.</td>
</tr>
</tbody>
</table>

If there are multiple AND or OR operators within a rule, AND operators take precedence over OR operators. This means that all AND operations in the rule are performed first. Only then are the OR operations considered. Thus, logic such as this:
IF  Amount  \leq  5000 \\
AND  CreditRating  >  5 \\
OR  CurrentCustomer  =  1 \\
AND  CreditRating  \geq  5 \\

is handled as follows:

IF  (Amount  \leq  5000  AND  CreditRating  >  5) \\
OR  (CurrentCustomer  =  1  AND  CreditRating  \geq  5)

For this example, the **Expression Editor** dialog appears as follows:

The preceding rule is true if anyone requests an amount less than 5,000 and has a credit rating better than average. The rule is true for any current bank customer requests any amount and has a credit rating greater than or equal to the average. Both conditions may be true, or only one or the other of them may be true. If both conditions are false, then the rule (as a whole) is false.

In detail, the preceding rule works as follows:

1. This AND expression:

   IF  Amount  \leq  5000 \\
   AND  CreditRating  >  5

   Gives a result, true or false. Call this result “SafeBet.”

2. This AND expression:

   IF  CurrentCustomer  =  1 \\
   AND  CreditRating  \geq  5

   Gives a result, true or false. Call this result “KnownEntity.”

3. Once the AND operations in the rule have completed, the OR operation begins, as follows:

   IF  SafeBet is true \\
   OR  KnownEntity is true

4. From what we know about OR logic, we know that this rule (as a whole) is true if the customer is a SafeBet but not a KnownEntity, or if the customer is not a SafeBet but is a KnownEntity. Additionally, this rule is true if the customer is both a SafeBet and a KnownEntity.
3.3.4.2 Expression Values

Within a condition or the assign, return, or trace actions, the values can be any of the following items:

- A numeric value (integer or decimal), such as 1.1 or 23000.
- A string value, which must be enclosed in double quotes: "NY"
- If your production invokes the rule from a BPL business process using the <rule> element, you can specify a property in the general-purpose, persistent variable context, which is defined using the <context> and <property> elements in BPL. A property name is case-sensitive, and must not be enclosed in quotes, as in:
  
  PlaceOfBirth

- An expression using various permitted operators, literal values, and properties of context, as for example:

  \[
  \frac{(2+2) \times 5}{154.3} \\
  "hello" \& "world" \\
  Age \times 4 \\
  ((x=1) \lor (x=3)) \land (y=2))
  \]

- A built-in Ensemble function such as Min(), Max(), Round(n,m), or SubString(). The function name must include parentheses. It must also include any input parameters, such as the numeric values n and m for Round. If there are no input values for the function, then the open and close parentheses must be present, but empty.

- The Document variable, which represents the message object.

When you want to insert a value into a field on the rule set tab, you may type it into the text box directly. If the syntax is incorrect or inappropriate for the type of data expected for that field, you see the red error symbol when you try to save your changes.

If you correctly entered a Context Class in your rule definition, when you select a property in the Expression Editor, the text box provides choices of properties from the business process execution context of the identified BPL business process Context Class.

3.3.4.3 Expression Operators

In the Expression Editor, you can click \( \bigoplus \) to choose from a list of operators of which to make the selected node an operand.

You may use the following arithmetic operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Plus (binary and unary)</td>
</tr>
<tr>
<td>–</td>
<td>Minus (binary and unary)</td>
</tr>
<tr>
<td>*</td>
<td>Times</td>
</tr>
<tr>
<td>/</td>
<td>Divide</td>
</tr>
</tbody>
</table>

You may use the following logical operators, which return an integer value of 1 (true) or 0 (false):

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Expression is true when...</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND (&amp;&amp;)</td>
<td>And</td>
<td>Both values are true.</td>
</tr>
<tr>
<td>OR (</td>
<td></td>
<td>)</td>
</tr>
<tr>
<td>Operator</td>
<td>Meaning</td>
<td>Expression is true when...</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>!</td>
<td>Not (unary)</td>
<td>The value is false.</td>
</tr>
<tr>
<td>=</td>
<td>Equals</td>
<td>The two values are equal.</td>
</tr>
<tr>
<td>!=</td>
<td>Does not equal</td>
<td>The two values are not equal.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Is greater than</td>
<td>The value to the left of the operator is greater than the value to the right of the operator.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Is less than</td>
<td>The value to the left is less than the value to the right.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Is greater than or equal to</td>
<td>The value to the left is greater than the value to the right, or if the two values are equal.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Is less than or equal to</td>
<td>The value to the left is less than the value to the right, or if the two values are equal.</td>
</tr>
<tr>
<td>[</td>
<td>Contains</td>
<td>The string contains the substring to the right. Pattern matching for Contains is exact. If the value at left is “Hollywood, California” and the value at right is “od, Ca”, there is a match, but a value of “Wood” does not match.</td>
</tr>
</tbody>
</table>

You may use the following string operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>Concatenation operator for strings.</td>
</tr>
<tr>
<td>_</td>
<td>Binary concatenation to combine string literals, expressions, and variables.</td>
</tr>
</tbody>
</table>

When more than one operator is found in an expression, the operators are evaluated in the following precedence order, from first to last:

1. Any of these logical operators: ! = != < > <= >=
2. Multiplication and division: * /
3. Addition and subtraction: + –
4. String concatenation: &
5. Logical AND: &&
6. Logical OR: ||

### 3.3.4.4 Expression Functions

Within a rule definition, an expression can include a call to one of the Ensemble utility functions. These include mathematical or string processing functions such as you may be accustomed to using in other programming languages.

In the Expression Editor, you can click ![function icon] to choose from a list of functions of which to make the focused node an argument.

For a list of the available utility functions and the proper syntax for using them in business rules or DTL data transformations, see “Ensemble Utility Functions” in Developing Business Rules.

### 3.3.4.5 Expression Examples

Within a rule definition, an expression is a formula for combining values and properties to return a value. For example:
<table>
<thead>
<tr>
<th>Expression</th>
<th>Computed value</th>
</tr>
</thead>
<tbody>
<tr>
<td>((2+2)\times5/154.3)</td>
<td>0.129617628</td>
</tr>
<tr>
<td>&quot;hello&quot; &amp; &quot;world&quot;</td>
<td>&quot;helloworld&quot;</td>
</tr>
<tr>
<td>Age * 4</td>
<td>When Age is a context property (a property in the general-purpose, persistent variable context, which is defined using the &lt;context&gt; and &lt;property&gt; elements in BPL) and has the numeric value 30, the value of this expression is 120.</td>
</tr>
<tr>
<td>1+2.5*2</td>
<td>6</td>
</tr>
<tr>
<td>2*5</td>
<td>10</td>
</tr>
<tr>
<td>Min(Age,80,Limit)</td>
<td>This expression uses the built-in function Min. When Age is a context property with the value 30 and Limit (likewise a property) has the value 65, the value of this expression is 30.</td>
</tr>
<tr>
<td>Round(1/3,2)</td>
<td>0.33. This expression uses the built-in function Round.</td>
</tr>
<tr>
<td>x&lt;65&amp;&amp;A=&quot;F&quot;</td>
<td></td>
</tr>
<tr>
<td>Min(10,Max(X,Y))</td>
<td>This expression uses the utility functions Min and Max. When X is a context property with the numeric value 9.125, and Y (likewise a property) has the numeric value 6.875, the value of this expression is 9.125.</td>
</tr>
<tr>
<td>(((x=1)</td>
<td></td>
</tr>
</tbody>
</table>

When you select a property that takes an expression as its value, a blank text field displays at the top of the rule set diagram. You may type any string in this field, so take care to enter the correct syntax. The rules for formulating expressions are as follows:

- An expression may involve any values as described in the preceding topics: numbers, strings, context properties, expressions, functions, or any valid combination of these.
- White space in expressions is ignored.
- You may use any of the described operators.
- If you want to override the default operator precedence, or if you want to make expressions easier to read, you can indicate precedence by using parentheses to group parts of the expression. Thus:
  
  \[1+2.5\times2 = 6\]
  
  Whereas:
  
  \[(1+2.5)\times2 = 7\]
- Business rules support parentheses to group complex logical expressions such as \(((x=1) \mid (x=3)) \&\& (y=2))\.

When you invoke the Expression Editor, a blue bar displays above the graphical representation of the expression that contains the textual representation of the expression.
3.4 Adding Business Rule Notification

It is possible for you to set up rule notification, so that specific actions are taken each time a rule is fired. Unlike most activities related to rules, notification requires programming. You must subclass Ens.Rule.Notification and override the %OnNotify method in the subclass. The signature of this method is:

```
ClassMethod %OnNotify(pReason As %String,
                       pRule As Ens.Rule.RuleDefinition)
                         As %Status
```

Possible `pReason` values are:

- `BeforeSave`
- `AfterSave`
- `Delete`

At runtime, the Ensemble framework automatically finds your subclass of Ens.Rule.Notification and uses the code in %OnNotify to determine what to do upon firing a rule.
Example Rule Classes

Ensemble contains various demonstration productions that contain examples of the different rule types. The following sections display an example rule class for each type:

- General Business Rule Example
- HL7 Message Routing Rule Example
- General Message Routing Rule Example
- Virtual Document Message Routing Rule Example

4.1 General Business Rule Example


The following figure shows how the rule definition looks in the Ensemble Rule Editor:

![Rule Definition Diagram]

The following is the class definition:
4.2 HL7 Message Routing Rule Example

The Demo.HL7.MsgRouter.XYZRoutingRule class is an example of a HL7 message routing business rule. It is part of the Demo.HL7.MsgRouter.Production production in the ENSDEMO namespace.

The following figure shows how the rule definition looks in the Ensemble Rule Editor:
The following code shows the class definition:

```java
/// Routing from the XYZ message source.
Class Demo.HL7.MsgRouter.XYZRoutingRule Extends Ens.Rule.Definition {
  Parameter RuleAssistClass = "EnsLib.HL7.MsgRouter.RuleAssistCompatible";
  XData RuleDefinition [ XMLNamespace = "http://www.intersystems.com/rule" ] {
    <ruleDefinition alias="" context="EnsLib.HL7.MsgRouter.RoutingEngine">
      <ruleSet name="" effectiveBegin="" effectiveEnd="">
        <rule name="">
          <constraint name="msgClass" value="EnsLib.HL7.Message"></constraint>
          <constraint name="docCategory" value="Demo.HL7.MsgRouter.Schema"></constraint>
          <when condition="!(HL7.ParentId!=")">
            <send transform="" target="Batch_FileOperation"></send>
            <return></return>
          </when>
        </rule>
        <rule name="">
          <constraint name="msgClass" value="EnsLib.HL7.Message"></constraint>
          <constraint name="docCategory" value="2.3.1"></constraint>
          <constraint name="docType" value="FHS,BHS"></constraint>
          <when condition="1">
            <send transform="" target="Batch_FileOperation"></send>
            <return></return>
          </when>
        </rule>
        <rule name="">
          <constraint name="msgClass" value="EnsLib.HL7.Message"></constraint>
          <constraint name="docName" value="ADT_A08"></constraint>
          <constraint name="docCategory" value="Demo.HL7.MsgRouter.Schema"></constraint>
          <constraint name="docType" value="ADT_A01"></constraint>
          <when condition="!(Contains(HL7.{PID:PatientName(1).familylastname}," &lt;S&gt;))
            || (startsWith(HL7.{PID:PatientName(1).familylastname},"Z"))">
            <send transform="" target="Other_FileOperation"></send>
            <return></return>
          </when>
        </rule>
      </ruleSet>
    </ruleDefinition>
  }
}
```

Developing Business Rules
Example Rule Classes

```xml
<send transform="Demo.HL7.MsgRouter.ADTLastNameTransform" target="Other_FileOperation"></send>
<return/></return>
</when>
</rule>

<constraint name="msgClass" value="EnsLib.HL7.Message"></constraint>
<constraint name="docCategory" value="Demo.HL7.MsgRouter.Schema"></constraint>
<when condition="(StartsWith(HL7.{PIDgrp.PID:PatientName(1).familylastname},"S")
|| (StartsWith(HL7.{PIDgrp.PID:PatientName(1).familylastname},"Z"))">
<send transform="Demo.HL7.MsgRouter.ORMLastNameTransform" target="Other_FileOperation"></send>
<return/></return>
</when>
</rule>
</ruleSet>
</ruleDefinition>

4.3 General Message Routing Rule Example

The Demo.HL7.MsgRouter.AlertRule class is an example of a general message routing business rule. It is part of the Demo.HL7.MsgRouter.Production production in the ENSDEMO namespace.

The following figure shows how the rule definition looks in the Ensemble Rule Editor:

![Rule Definition Diagram]

The following code shows the class definition:

```xml
/// See the AlertTable lookup table for a mapping of alert sources to email addresses.
/// If no match is found in the lookup, EmailAlertTransformation sends the alert
/// to the configured Recipient for the EMailAlertOperation.

    Parameter RuleAssistClass = "EnsLib.MsgRouter.RuleAssist";

    XData RuleDefinition [ XMLNamespace = "http://www.intersystems.com/rule" ]
    <ruleDefinition alias="" context="EnsLib.MsgRouter.RoutingEngine">
        <ruleSet name="" effectiveBegin="" effectiveEnd="" />
        <constraint name="msgClass" value="Ens.AlertRequest"></constraint>
        <when condition="1">
            <send transform="Demo.HL7.MsgRouter.EmailAlertTransform" target="EMailAlertOperation"></send>
            <return/></return>
        </when>
    </rule>
```
4.4 Virtual Document Message Routing Rule Example

The Demo.HL7v3.Rule.RouteAndTransform class is an example of a virtual document message routing business rule. It is part of the Demo.HL7v3.Production.InterfaceEngine production in the ENSDEMO namespace.

The following figure shows how the rule definition looks in the Ensemble Rule Editor:

```
class Demo.HL7v3.Rule.RouteAndTransform Extends Ens.Rule.Definition
{
    Parameter RuleAssistClass = "EnsLib.MsgRouter.VDocRuleAssistCompatible";

    XData RuleDefinition [ XMLNamespace = "http://www.intersystems.com/rule" ]
    {
        <ruleDefinition alias="" context="Demo.HL7v3.Rule.Context">
            <ruleSet name="" effectiveBegin="" effectiveEnd="">
                <rule name="" disabled="false">
                    <constraint name="msgClass" value="Demo.HL7v3.Message"></constraint>
                    <constraint name="docName" value="MFMT_IN002101"></constraint>
                    <when condition="1">
                        <send transform="Demo.HL7v3.Transformation.MFMTIN002101" target="HL7v3 SOAP Out"></send>
                        <send transform="Demo.HL7v3.Transformation.MFMTIN002101" target="HL7v3 File Out MFMT"></send>
                        <return></return>
                    </when>
                </rule>
                <rule name="" disabled="false">
                    <constraint name="msgClass" value="Demo.HL7v3.Message"></constraint>
                    <constraint name="docName" value="QUPA_IN101103"></constraint>
                    <when condition="1">
                        <send transform="Demo.HL7v3.Transformation.QUPAIN101103" target="HL7v3 SOAP Out"></send>
                        <send transform="Demo.HL7v3.Transformation.QUPAIN101103" target="HL7v3 File Out QUPA"></send>
                        <return></return>
                    </when>
                </rule>
            </ruleSet>
        </ruleDefinition>
    }
}``
```
Example Rule Classes

]}</ruleDefinition>
}

}
This chapter describes strategies for debugging the routing rules in an EDI message routing production.

The primary symptom for problems in routing rules is that the message does not reach its destination. Perhaps the message reaches a point along the way, such as a business operation or routing process within the routing production, but it does not reach its target destination, which is generally an application server outside Ensemble. In that case you can follow the problem-solving sequence captured in the next four drawings: “Solving Problems with Routing Rules,” Drawings A, B, C, and D.
Figure 5-1: Solving Problems with Routing Rules (Drawing A)

**Problem:** My message does not arrive at its destination

1. **View the Visual Trace in the Message Browser**

   View message contents.

   **Does the message have a DocType and Message Schema Category?**
   
   - **NO**
   - **YES**

   **Did the message validate properly?**
   
   - **NO, it had BuildMapStatus errors**
   - **YES**

   **Likely Cause: Validation Error**

   - Check the Business Service in the Production Configuration. Does it have a Message Schema Category?
     
     - **NO** Configure it with one.
     - **YES**

   - Check for common problems:
     - The VDoc value specifies a virtual document type that does not exist
     - No DocType was specified (entire message contents are in black type without any color coding)

   - Analyze message contents for other causes of validation errors

   **Does the message go to the expected Operation?**
   
   - **NO, the message stops at the Message Router Process** [See Drawing B]
   - **YES, the message is sent to the expected Operation** [See Drawing D]
   - **NO, the message is sent to a different Operation** [See Drawing D]
Figure 5–2: Solving Problems with Routing Rules (Drawing B)

Problem: My message does not arrive at its destination

1. View the Visual Trace in the Message Browser
   
   Does the message go to the expected Operation?
   
   NO, the message stops at the Message Router Process

2. View the Business Rule Log
   
   Does an entry exist in the business rule log for this transaction?
   
   NO
   
   YES

   Is it an error? (Shown in red)
   
   NO ➔ See Drawing C
   
   YES, Rule Missing ➔ Likely Cause: Rule Naming Error

   
   Is your rule in the list of rules?
   
   NO
   
   YES ➔ Verify the rule name. The package and rule name are case-sensitive.

4. View Production Configuration
   
   The full Package.Name for this rule should be specified in the Business Rule Name field of the Message Router Process. Is the name there?
   
   Is it correct? (It is case-sensitive)
   
   Edit rule as needed to match names

   Is the problem fixed?
   
   NO

5. Check the Event Log
Figure 5–3: Solving Problems with Routing Rules (Drawing C)

Problem: My message does not arrive at its destination

2. *View the Business Rule Log*

Does an entry exist in the business rule log for this transaction?

- YES
  - Is it an error? (Shown in red)
    - NO
      - View the Reason and Return fields
    - YES
      - Are the Reason and Return fields empty?
        - NO
          - The rule found a match
            - Does the Result list the Operation?
              - YES
                - Is there a data transformation?
                  - NO
                    - *Likely Cause: Rule Definition Error*
                      - Check your rule for common problems:
                        - Are you using the correct rule name?
                        - Is the Source correct?
                        - Does the Schema Category in the rule match the Message Schema Category in the business service?
                        - Does the Schema DocType in the rule match the DocType in the message contents?
                        - View the Conditions and interpret these against the message payload in the Visual Trace.
                        - Does the rule refer to a property or function that does not exist?
                        - Does the rule refer to a user-defined function that contains an error?
                  - YES
                    - *Likely Cause: Transformation Error*
                      - Check for common problems:
                        - The specified Data Transformation does not exist or is misnamed
                        - There is a logical error inside the Data Transformation
                        - Check the Event Log to see what happened
                - NO
                  - 4. *Check the Event Log*

- YES
  - Your message did not match any rule

Drawing B

Drawing C
Figure 5–4: Solving Problems with Routing Rules (Drawing D)

Problem: My message does not arrive at its destination

1. **View the Visual Trace in the Message Browser**
   - Does the message go to the expected Operation?
     - NO, the message is sent to a different Operation
     - YES, the message is sent to the expected Operation
   - **Drawing D**

   **Likely Cause: Logical Error**

2. **View the Business Rule Log.**
   - Did you expect the rule to match multiples, but it only matched one?
     - NO
     - YES
     - Edit the rule.
     - Ensure that DoAll is selected.
   - **Drawing A**

   **Likely Cause: Configuration Error**

3. **View the Production Configuration**
   - Is the Operation enabled (white=enabled, gray=disabled)?
     - NO
     - YES
     - Does the Operation have an error (red indicator)?
       - NO
       - YES
   - **Check the Event Log**

   **Likely Cause: Configuration Error**

   - **View Message Trace to inspect the message Type, DocType, Contents.**
   - **Review your rules; logically analyze.**

4. **Check the Event Log**

   - **Check the Queue. You should see your message.**
   - **Enable the Operation.**

   **Verify the Operation settings.**
   - File Operation: File Path, File Name
   - TCP Operation: Verify the IP and Port.
   - Check the Queue; Is there a backlog?
Ensemble Utility Functions

This appendix describes the Ensemble utility functions that you can use in business rules and DTL data transformations. These include mathematical or string processing functions such as you may be accustomed to using in other programming languages. This appendix contains the following topics:

- Built-in Functions
- Syntax to Invoke a Function

To define your own functions, see “Defining Custom Utility Functions” in Developing Ensemble Productions.

A.1 Built-in Functions

The following lists the utility functions built into Ensemble.

Note: For boolean values, 1 indicates true and 0 indicates false.

Contains(val,str)
Returns 1 (true) if val contains the substring str; otherwise 0 (false).

ConvertDateTime (val,in,out,file)
Reads the input string val as a time stamp in in format, and returns the same value converted to a time stamp in out format. See “Time Stamp Specifications for Filenames” in Configuring Ensemble Productions.

The default for in and out is %Q. Any %f elements in the out argument are replaced with the file string. If val does not match the in format, out is ignored and val is returned unchanged.

CurrentDateTime(format)
Returns a string representing a date/time value in the given format. The default format is ODBC format (%Q) in the server’s local timezone.

DoesNotContain(val,str)
Returns 1 (true) if val does not contain the substring str.
DoesNotIntersectList(val, items, srcsep, targetsep)

Returns 1 (true) if no item in the given source list (val) appears in the target list (items). For details on the arguments, see IntersectsList.

DoesNotMatch(val, pat)

Returns 1 (true) if val does not match the pattern specified by pat. pat must be a string that uses syntax suitable for the ? pattern matching operator in ObjectScript. For details, see “Pattern Matching” in the chapter “Operators and Expressions” of Using Caché ObjectScript.

DoesNotStartWith(val, str)

Returns 1 (true) if val does not start with the substring str.

Exists(tab, val)

The Exists function provides a way to predict the results of the Lookup function. Exists returns 1 (true) if val is a key defined within the table identified by tab; otherwise it returns 0 (false).

If(val, true, false)

If the argument val evaluates to 1 (true), the If function returns the string value of its true argument; otherwise it returns the string value of its false argument.

In(val, items)

Returns 1 (true) if val is found in the comma-delimited string items.

InFile(val, file)

Returns 1 (true) if val is found in the identified file.

InFileColumn(...)

The function InFileColumn can have as many as 8 arguments. The full function signature is:

InFileColumn(val, file, columnId, rowSeparator, columnSeparator, columnWidth, lineComment, stripPadChars)

InFileColumn returns 1 (true) if val is in the specified column in a table-formatted text file. Arguments are as follows:

• val (required) is the value.
• file (required) is the text file.
• Default columnId is 1.
• Default rowSeparator is ASCII 10. A negative rowSeparator value indicates the row length.
• Default columnSeparator is ASCII 9. If columnSeparator is 0, the format of the file is said to be “positional.” In this case columnId means character position and columnWidth means character count.
• Default columnWidth is 0.
• Default lineComment is an empty string.
• Default stripPadChars consists of a blank space followed by ASCII 9.

IntersectsList(val, items, srcsep, targetsep)

Returns 1 (true) if any item in the given source list (val) appears in the target list (items). The arguments srcsep and targetsep specify the list separators in the source and target lists respectively; for each of these, the default is "><", which means that the lists are assumed to have the form "<item1><item2><item3>"
if the target list has only a single item, this function is essentially the same as the Contains function. If the source list has only a single item, this function is essentially the same as the In function.

**Length(string,delimiter)**

Returns the length of the given string. If you specify delimiter, this function returns the number of substrings based on this delimiter.

**Like(string,pattern)**

Returns 1 (true) if the given value (string) satisfies a SQL Like comparison with the given pattern string (pattern). In SQL Like patterns, % matches 0 or more characters, and _ matches any single character. Note that an escape character can be specified by appending “%%” to the pattern, e.g. "##SYSVAR_#_%%##" to match any value string that starts with "%%SYSVAR" followed by any single character, an underscore, and anything else.

**Lookup(table,keyvalue,default,defaultOnEmptyInput)**

The **Lookup()** function searches for the key value specified by keyvalue in the table specified by table and returns its associated value. This returned value is equivalent to the following global:

```
^Ens.LookupTable(table,keyvalue)
```

If the key is not found in the table, the Lookup function returns the default value specified by the default parameter. The default parameter is optional, so if it is not specified and Lookup does not find a matching key, it returns an empty string. An exception is that if either the key value or the lookup table is empty, the Lookup() function returns either the default value or the empty string depending on the value of the defaultOnEmptyInput parameter as described in the following table. The default value of the defaultOnEmptyInput parameter is 0 for compatibility with previous versions of Ensemble.

<table>
<thead>
<tr>
<th>defaultOnEmptyInput Value</th>
<th>key value and lookup table</th>
<th>Lookup() returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>either key value or lookup table is empty</td>
<td>empty string</td>
</tr>
<tr>
<td>1</td>
<td>key value is empty</td>
<td>empty string</td>
</tr>
<tr>
<td></td>
<td>lookup table is empty but key value is not</td>
<td>default value</td>
</tr>
<tr>
<td>2</td>
<td>lookup table is empty</td>
<td>empty string</td>
</tr>
<tr>
<td></td>
<td>key value is empty but lookup table is not</td>
<td>default value</td>
</tr>
<tr>
<td>3</td>
<td>either key value or lookup table is empty</td>
<td>default value</td>
</tr>
</tbody>
</table>

The **Exists()** function returns true if a **Lookup()** function with the same parameters would find the key value in the lookup table.

**Matches(val,pat)**

Returns 1 (true) if val matches the pattern specified by pat. pat must be a string that uses syntax suitable for the ? pattern matching operator in ObjectScript. For details, see “Pattern Matching” in the chapter “Operators and Expressions” of Using Caché ObjectScript.
Max(...)

Returns the largest of a list of up to 8 values. List entries are separated by commas.

Min(...)

Returns the smallest of a list of up to 8 values. List entries are separated by commas.

Not(val)

Returns 0 (false) if val is 1 (true); 1 (true) if val is 0 (false).

NotIn(val,items)

Returns 1 (true) if val is not found in the comma-delimited string items.

NotInFile(val,file)

Returns 1 (true) if val is not found in the identified file.

NotLike(string,pattern)

Returns 1 (true) if the given value (string) does not satisfy a SQL Like comparison with the given pattern string (pattern). See Like.

Pad(val,width,char)

Reads the input string val. Adds enough instances of char to widen the string to width characters. If width is a positive value, the padding is appended to the right-hand side of the val string. If width is a negative value, the padding is prepended to the left-hand side of the val string.

Piece(val,char,from,to)

If the delimiter character char is present in the string val, this separates the string into pieces. If there are multiple pieces in the string, from and to specify which range of these pieces to return, starting at 1. If multiple pieces are returned, the delimiter in the return string is the same as the delimiter in the input string. For example:

Piece("A,B,C,D,E,F") returns "A"

Piece("A!B!C!D!E!F","!",2,4) returns "B!C!D"

The default char is a comma, the default from is 1, and the default to is from (return one piece). For details, see the $PIECE function in the Caché ObjectScript Reference.

ReplaceStr(val,find,repl)

Reads the input string val. Replaces any occurrences of string find with the string repl, and returns the resulting string.

Note: Use ReplaceStr instead of the Replace function, which has been deprecated.

Round(val,n)

Returns val rounded off to n digits after the decimal point. If n is not provided (that is, Round(val)) the function drops the fractional portion of the number and rounds it to the decimal point, producing an integer.

Rule(rulename,context,activity)

Evaluates the rule specified in the rulename with the given context object and the given activity label for the Rule Log and returns the value.
Schedule(ScheduleSpec, ODBCDateTime)

Evaluates the state of the given ScheduleSpec string, named Schedule or Rule at the moment given by ODBCDateTime. If ScheduleSpec begins with '@' it is a Schedule name or Rule name, otherwise a raw Schedule string. If ODBCDateTime is blank, the evaluation is done for the current time.

StartsWith(val, str)

Returns 1 (true) if val starts with the substring str.

Strip(val, act, rem, keep)

Reads the input string val. Removes any characters matching the categories specified in the act template and the rem string, while retaining any characters found in the keep string. Returns the resulting string. For details and examples, see the $ZSTRIP function in the Caché ObjectScript Reference.

SubString(str, n, m)

Returns a substring of a string str, starting at numeric position n and continuing until numeric position m. The number 1 indicates the first character in the string. If m is not provided (that is, SubString(str,n)) the function returns the substring from position n to the end of the string.

ToLower(str)

Returns the string str converted to lowercase.

ToUpper(str)

Returns the string str converted to uppercase.

Translate(val, in, out)

Reads the input string val. Translates each occurrence of a character in string in to the character at the corresponding position in string out, and returns the resulting string.

Note: These functions are defined by methods in the class Ens.Util.FunctionSet.

A.2 Syntax to Invoke a Function

When you reference an Ensemble function in a business rule or a DTL data transformation, the syntax must include parentheses. It must also include any input parameters, such as the numeric values for the mathematical functions Min, Max, or Round. If there are no input values for the function, then the open and close parentheses must be present, but empty.

The following are examples of valid function syntax:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Computed Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min(Age, 80, Limit)</td>
<td>When Age is a property with the value 30 and Limit (likewise a property) has the value 65, the value of this expression is 30.</td>
</tr>
<tr>
<td>Round(1/3, 2)</td>
<td>0.33</td>
</tr>
<tr>
<td>Min(10, Max(X,Y))</td>
<td>When X is a property with the numeric value 9.125, and Y (likewise a property) has the numeric value 6.875, the value of this expression is 9.125.</td>
</tr>
</tbody>
</table>
If the value input to any function is a string that starts with a number, nonnumeric characters in the string are dropped and the numeric portion is used. The string "3a" is treated like the number 3, so the function \text{Min}("3a","2OfThem") returns the value 2. A string that begins with a nonnumeric character such as "a123" has the numeric value 0.

The business rule syntax for utility functions differs from the DTL syntax in the following significant way:

- Business rules reference the utility functions simply by name:
  \text{ToUpper}(value)

- DTL uses two leading dots immediately before the function name, as if the function were a method:
  ..ToUpper(value)

The following DTL data transformation uses a utility function called \text{ToUpper()} to convert a string to all uppercase characters. The \text{<assign>} statement references \text{ToUpper()} using double-dot syntax, as if it were a method in the class:

```xml
Class User.NewDTL1 Extends Ens.DataTransformDTL {

XData DTL {
  <?xml version="1.0" ?>
  <transform targetClass='Demo.Loan.Msg.Approval' sourceClass='Demo.Loan.Msg.Approval' create='new' language='objectscript'>
    <assign property='target.BankName' value='..ToUpper(source.BankName)' action='set'/>
    <trace value='Changed all lowercase letters to uppercase!' />
  </transform>
}
}
```