COMPUWARE

Compuware
Optimal Trace™

Optimal Trace™ - Customizing Exports & Reports with Velocity and the Optimal Trace API

August 2007

1 Introduction

Optimal Trace provides an in-built ability to generate web-based reports and structured text exports of all Optimal Trace project data.

This guide covers:

- ❑ how text exports work in Optimal Trace
- ❑ how web reports work in Optimal Trace
- ❑ how to introduce customised reports and exports and
- ❑ the full Optimal Trace API

Readers of this user guide will be:

- Those who would like to understand the mechanics of how the current web based reporting and project export facilities works.
- Those who would like to understand how to build custom web reports for publication.
- Those who would like to understand how to build general text exports for usage in excel as CSV files etc.
- Those who would like to understand how to build text exports specifically to 3rd party environments that support imports such as text or XMI in the case of UML supporting tools.

Note 1: It is highly recommended that readers interested in creating their own custom reports or exports should have a basic technical knowledge of scripting languages and specifically the Velocity Template Engine from the Apache Software Foundation.

Note 2: This version of the user guide has been updated to reflect Optimal Trace 5.0 or later (both Enterprise and Professional). Previous versions of Optimal Trace have differences in terms of the API and directory structure. You should upgrade to 5.0 or later prior to trying any of the examples in this paper.

2 The Structure of Optimal Trace Exports (Export Profiles)

Optimal Trace provides an ability to export project data to a variety of text formats.

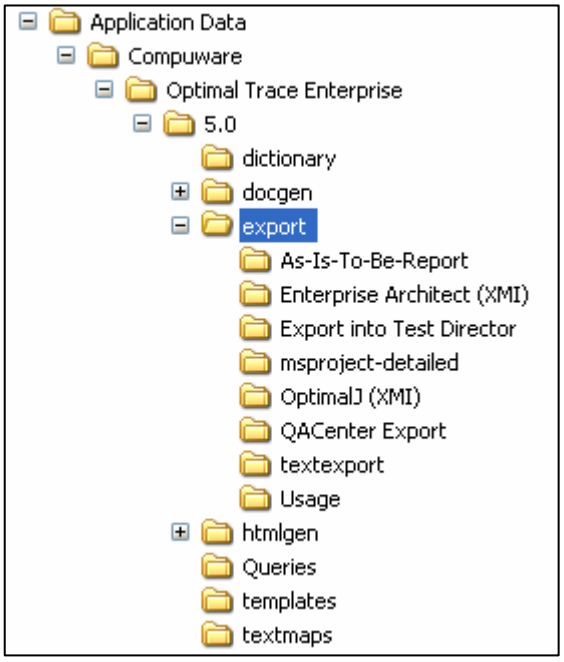
The following exports ship with Optimal Trace:

- Text Export
- MS Project Export
- Text Actor Usage Export
- Text As Is - To Be Export
- Optimal J XMI (UML)
- Test Director Export

Note: Full details of each of these Exports can be found in section: 12 Appendix 3: Detailed List of all text exports and reports.

To run these exports in Optimal Trace:

- From the Menu bar, click on Project
- Click on Export Project…
- Select the profile you wish to use for report generation.
- Click on the Export button.



Optimal Trace uses an **Export Profile** for each export. Export Profiles dictate the make up of the export.

Each Export Profile comprises the following components:

- a profile.xml file
- one or more script files (referred to also as a template)

The profile.xml contains the 'directives' that define the export. It specifies aspects such as where the export will be generated to (i.e. the output directory), what name the export will have when showing within Optimal Trace and what script file(s) will be used for generation.

For each export there will always be one profile.xml file and at least one script file. Each export is contained in a sub-directory under the location:

X:\Documents and Settings\<user name>\Application Data\Compuware\Optimal Trace\5.0\export.

- where X is the default user drive (Usually C:) and <user name> is the login name of the user.

It should be noted that every user on a machine gets their own copy of the exports directory. Where relevant, this is called the personal exports directory in the remainder of this document.

3 The Structure of Optimal Trace Web Reports (Report Profiles)

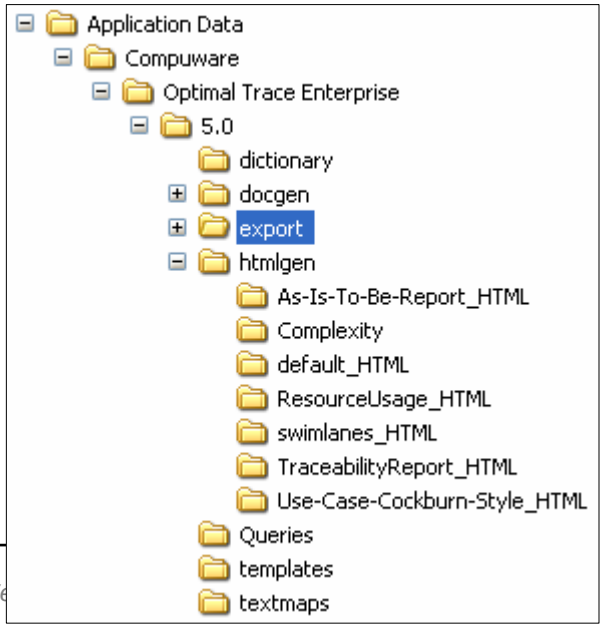
Similar to the export mechanism, Optimal Trace ships with an ability to generate web-based reports. Out of the box, the following reports area available:

- General Report (default_HTML)
- Swimlane Report (swimlanes_HTML)
- Requirement (AC) Report (Use-Case-Cockburn-Style_HTML)
- Actor Usage Report (ResourceUsage_HTML)
- Traceability Report (TraceabilityReport_HTML)
- As Is - To Be Report(As-Is-To-Be-Report_HTML)

Note: Full details of each of these Exports can be found in section: 12 Appendix 3: Detailed List of all text exports and reports.

To view these reports running on a given project in Optimal Trace:

- From the Menu bar, click on Generation
- Click on Generate Reports…
- Select the profile you wish to use for report generation.
- Click on the Generate button.



Underlying each report is a **Report Profile.** Report Profiles dictate the makeup of the report. Each Report Profile comprises the following components:

- A profile.xml file
- One or more script files (referred to as a template)
- Any additional files used in the HTML report (e.g.: stylesheets, *.css files, images etc.)

The profile.xml contains the 'directives' that define the report. It specifies aspects such as where the

report will be generated to (i.e. the output directory), what name the report will have when showing within Optimal Trace, what script file(s) will be used for generation and what additional files, if any, to copy to the output directory. (each of these settings are explained in detail in section:7 The Structure of the Profile.xml file)

For every report profile there will always be one profile.xml file, at least one script file and possibly a set of additional files (jpg etc.).
Each report profile is contained in a sub-directory under the location:

X:\Documents and Settings\<user name>\Application Data\Compuware\Optimal Trace\5.0\htmlgen

- where X is the default user drive (Usually C:) and <user name> is the login name of the user.

-

It should be noted that every user on a machine gets their own copy of the HTML gen directory. In the remainder of this document, this directory is referred to as the htmlgen directory.

Note Since Reports are HTML based, there is often a need to copy over additional files (such as CSS or graphic files) for the report. As exports are text based there is no need to copy over additional files to support the export process.

4 How Exporting and Report Generation work

Let's now consider the script files that form the basis of what is generated for both reports and exports.

Optimal Trace ships with a scripting language called Velocity that together with the Optimal Trace API (Application Programming Interface), forms the core of every Optimal Trace Report or Export script file.

Optimal Trace Reports are in HTML format. The specific script files are actually HTML templates with embedded velocity script between HTML tags.

A Note on Velocity: Optimal Trace uses the Velocity Template Engine from the Apache Software Foundation. It is recommended that anybody who intends writing custom Optimal Trace text generation templates first familiarise themselves with Velocity, and has a working knowledge of Java in order to read the Optimal Trace API documentation. For a very good introduction to Velocity, see this article on JavaWorld.

For example, a sample report snippit follows:

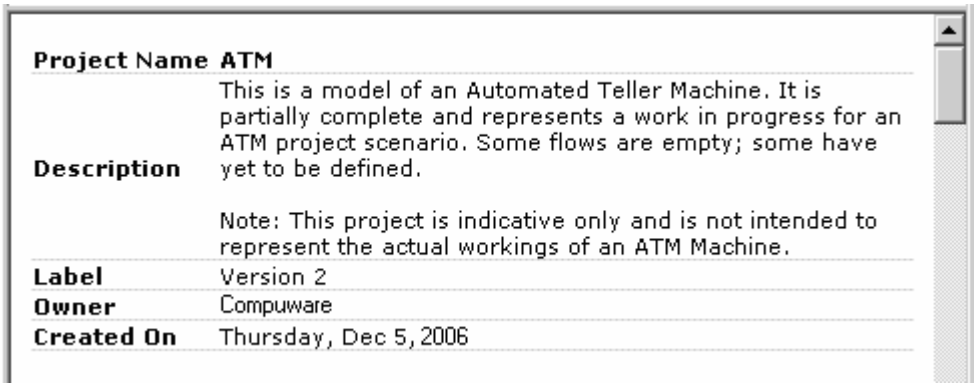
```
1. <table  width="100%"  border="0" cellspacing="1" class="containment-border">
2. <tr>
3. <td width="21%" ><b>Project Name</b></td>
4. <td width="79%" ><b>#escapeChars($project.Name)</b></td>
5. </tr>
6. <tr>
7. <td width="21%" ><b>Description</b></td>
8. <td width="79%" >#escapeChars($project.LongDescription)</td>
9. </tr>
10. <tr>
11. <td width="21%" ><b>Label</b></td>
12. <td width="79%" >#escapeChars($project.VersionLabelName)</td>
13. </tr>
14. <tr>
15. <td width="21%" ><b>Owner</b></td>
```

```
16. <td width="79%" >#escapeChars($project.Owner)</td>
17. </tr>
18. <tr>
19. <td width="21%" ><b>Created On</b></td>
20. <td width="79%" >#escapeChars($project.CreationDateAsString)</td>
21. </tr>
22. </table>
```

This extract is taken from the script file 'ProjectDetail.HTML' that ships with the 'General Report', in Optimal Trace Enterprise and Professional. (See ..\HTMLgen\default_HTML\projectdetail.HTML).

As can be seen in this example, the HTML has embedded velocity script. Line 4 is an instance of a velocity macro being called with an argument *$project.Name* that is coming from the Optimal Trace API. Lines 8, 12, 16 and 20 similarly show other attributes of the Optimal Trace project passed as arguments to the velocity macro *#escapeChars*. This macro strips out any characters that would be illegal in the context of well formed HTML output and outputs the result.

The web based HTML output created from this report appears as follows:



Project Name	**ATM**
Description	This is a model of an Automated Teller Machine. It is partially complete and represents a work in progress for an ATM project scenario. Some flows are empty; some have yet to be defined. Note: This project is indicative only and is not intended to represent the actual workings of an ATM Machine.
Label	Version 2
Owner	Compuware
Created On	Thursday, Dec 5, 2006

For exports, the mechanics are very similar, the only difference being that the velocity script does not sit within any HTML tags. Optimal Trace exports are therefore simply text files containing velocity script.

Custom text exports can be used to generate just about any form of text file that you think might be useful given the content of a Optimal Trace project. Similarly, an unlimited variety of web reports can be generated using Reports Profiles.

For example, from a text export perspective, you could feasibly generate java test code directly from the content of each Requirement. From a report perspective you could generate a HTML report that filters the Requirements in a Project by the values contained within certain Custom Properties. See section: 6.3 Example 3: Displaying Increments of Delivery - filtering on Custom Property for a good example of this approach. This 'Delivery Report' displays all Requirements that are planned for a certain iteration (or release). The report filters by a custom property called 'Increment'.

Using the mechanism of Custom Report Profiles or Export Profiles, you can insert your own customized Reports or Exports into the Optimal Trace environment. The next section will demonstrate this.

5 Optimal Trace Exports

Optimal Trace uses a generic mechanism for finding new Exports and Reports. On start up, it will search the personal export directory and if it sees a suitable Export Profile, it will dynamically populate the Export List in the tool.

By simply adding to this folder a directory containing your Export files and starting Optimal Trace, the new Export will automatically be added to the list. Lets look at an example export.

5.1 Example: A Simple Text Export

As a simple example, lets suppose we want to generate an export (as a text file), containing the Project Name and a list of all the names of each Requirement in the Project.
Firstly, using any standard text editor (notepad etc.), create a new text file. Call this file 'Requirement-List.txt'. Now we'll add some velocity script that will drive the output created by the export.

Note on the Optimal Trace API:
Optimal Trace speaks in terms of Requirements, with each Requirement comprised of a set of scenarios and steps. From the API perspective, each Requirement is termed a 'UseCase'. This stems from early versions of Optimal Trace that spoke in terms of use cases. With later releases of Optimal Trace this notion has been expanded to include other non-use case specific aspects.

Now add the following lines to the script file:

```
Project: $project.getName()
List of Requirements:
#foreach ($usecase in $project.getUseCasePackage().getAllUseCasesInPackages())
Name - $usecase.getDisplayName()
#end
```

This will produce the following text output for the Optimal Trace ATM demo:

```
Project: ATM
List of Requirements:
Name - SR1: Use ATM Machine
Name - SR2: Withdraw Cash
Name - SR3: Deposit Cash
Name - SR4: Transfer Funds
Name - SR5: Check Balance
```

Lets break this down line by line.

Line one:

```
Project: $project.getName()
```

'Project:' is free form text. '$project' refers to the Optimal Trace API Project object(discussed in section 9 The Optimal Trace API), and 'getName()' is the operation to call on the Project.

Line two:

List of Requirements:

This again is simply free form text. Since there are no velocity instructions it is output as it appears.

Line three:

#foreach ($usecase in $project.getUseCasePackage().getAllUseCasesInPackages())

"project.getUseCasePackage().getAllUseCasesInPackages()" -gets a list of all the use cases in this project.

In short, this iterates through all the UseCase objects in the Project. '#foreach' is a Velocity keyword that creates a loop, '$usecase' establishes a Velocity variable called '$usecase' which is assigned the value of each successive element in the list as the loop progresses. We can use '$usecase' to refer to the Use Case object being iterated over, as shown in Line Four.

See the Optimal Trace API documentation for full details on how to access internal Optimal Trace objects.

Line four:

Name - $usecase.getDisplayName()

'Name –' is free form text. '$usecase' is the Velocity variable for the Use Case that we're currently iterating over. 'getDisplayName()' is the Optimal Trace API call on a UseCase object to get its display name.

Line five:

#end

'#end' is simply a Velocity keyword to close the corresponding '#foreach' iteration.

Overall:

This script displays the project name then uses '#foreach' to cycle through each element of the list returned by 'project.getUseCasePackage().getAllUseCasesInPackages()'. On each cycle through it displays the String returned by "$usecase.getDisplayName()".

That is a very simple export, however the Optimal Trace API is quite comprehensive and allows full access to any internal Optimal Trace objects that Optimal Trace itself uses, therefore, any of these objects can be used with a velocity script allowing you to generate very customizable exports. Additionally, velocity script allows us to interrogate the state of the model and code decision logic based on the state.

For detailed examples of Velocity templates using the Optimal Trace API, see the files that Optimal Trace itself uses, these will be located under the directory: <Optimal Trace installation directory>\export directory.

To add our example above to the Optimal Trace tool, we need to follow these steps:

1. Create a new directory called 'Basic Export' under the Optimal Trace 'export' directory.
2. Place the template file (Requirement-List.txt) created in the example above in this directory.

3. From the ‘export\usage’ directory, copy the file ‘profile.xml’ to this new ‘Basic Export’ directory. (We do this for convenience and we will modify that file. This export profile has only one script file and therefore will be quite similar to what we need)

As mentioned earlier each ‘export’ directory has a profile.xml file that contains configuration information for that export. Additionally each profile can contain one or more templates with each template corresponding to a text generation file (script file).

Take the profile.xml file and open it in your text editor. You will see the following (or similar):

(Comments are formed in a *.xml file with the tags <!--to open the comment and --> to close them.)

Note on Ids & Timestamps in the Profiles:
Optimal Trace profiles contain an Id entry of type long. Although you must ensure the id is present the actual value simply needs to be unique within the scope of the profile.
Timestamp entries are also present. This is legacy and is retained for backward compatibility.
Copying and adjusting existing profile.xml files is the easiest way to start a new profile.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <TextGenProfile Id="ST282810241458958" TimeStamp="1014065346295">
3. <DynAttributes>
4. <DynAttribute Name="Position" TimeStamp="1014065346295" Type="java.lang.Integer" Value="3"/>
5. <DynAttribute Name="Description" TimeStamp="1014065346275" Type="java.lang.String"
   Value="Export to CSV format with Actor usage content."/>
6. <DynAttribute Name="Name" TimeStamp="1014065346275" Type="java.lang.String" Value="Text
   Actor Usage Export"/>
7. <DynAttribute Name="isReadOnly" TimeStamp="1014065346275" Type="java.lang.Boolean"
   Value="false"/>
8. <DynAttribute Name="isLocked" TimeStamp="1014065346275" Type="java.lang.Boolean"
   Value="false"/>
9. <DynAttribute Name="OutputDirectory" TimeStamp="1014065346295" Type="java.lang.String"
   Value=""/>
10. <DynAttribute Name="OutputFileName" TimeStamp="1014065346295" Type="java.lang.String"
    Value="ActorUsage.csv"/>
11. <DynAttribute Name="ExternalLinkName" TimeStamp="1014065346295" Type="java.lang.String"
    Value="Actor Useage Report - TEXT"/>
12. </DynAttributes>
13. <TextGenTemplate Id="ST282810400007383" TimeStamp="1014065346285">
14. <DynAttributes>
        a. <DynAttribute Name="Name" TimeStamp="1014065346285" Type="java.lang.String"
           Value="TextReport"/>
        b. <DynAttribute Name="IsActivityDiagramNeeded" TimeStamp="1014065346285"
           Type="java.lang.Boolean" Value="false"/>
        c. <DynAttribute Name="IsUseCaseDiagramNeeded" TimeStamp="1014065346285"
           Type="java.lang.Boolean" Value="false"/>
        d. <DynAttribute Name="Description" TimeStamp="1014065346285" Type="java.lang.String"
           Value="CSV file template for Project"/>
        e. <DynAttribute Name="OutputFileName" TimeStamp="1014065346285"
           Type="java.lang.String" Value="ActorUsage.csv"/>
        f. <DynAttribute Name="AdditionalFilesToCopy" TimeStamp="1014065346285"
           Type="java.lang.String" Value=""/>
        g. <DynAttribute Name="isLocked" TimeStamp="1014065346285" Type="java.lang.Boolean"
           Value="false"/>
        h. <DynAttribute Name="ContextObject" TimeStamp="1014065346285"
           Type="java.lang.String" Value="Project"/>
        i. <DynAttribute Name="TemplateFileName" TimeStamp="1014065346285"
           Type="java.lang.String" Value="export/Usage/Usage.vm"/>
        j. <DynAttribute Name="ContextVariableName" TimeStamp="1014065346285"
           Type="java.lang.String" Value="project"/>
        k. <DynAttribute Name="isReadOnly" TimeStamp="1014065346285" Type="java.lang.Boolean"
           Value="false"/>
15. </DynAttributes>
```

16. </TextGenTemplate>
17. </TextGenProfile>

Using the mechanism of Custom Report Profiles or Export Profiles, you can insert your own customized Reports or Exports into Optimal Trace. The next section will demonstrate this.

There are two core XML areas of this file, the first is the <TextGenProfile> node, the second the <TextGenTemplate> node.

<TextGenProfile> controls aspects such as how this report will surface in Optimal Trace, while <TextGenTemplate> points at the specific velocity template containing the script.

Set the following attribute values for the <TextGenProfile> node:

- Line 4 set "Position" = "7"

  (the position the export will appear in the list of available exports)

- Line 5 set "Description" = "This profile will generate a list of all Requirement Names in a Project."

  (the export description as it appears in Optimal Trace)

- Line 6 set "Name" = " Basic Export "

  (the export name as it appears in Optimal Trace)

- Line 10 set "OutputFileName" = "Requirement-List.txt "

  (refers to the name of the generated file that will launch when hitting the 'Open' button in Optimal Trace at conclusion of the export.)

- Line 11 set "ExternalLinkName" = "Requirements List "

  (the name of the link if you check the 'Add Link to Project' option on export)

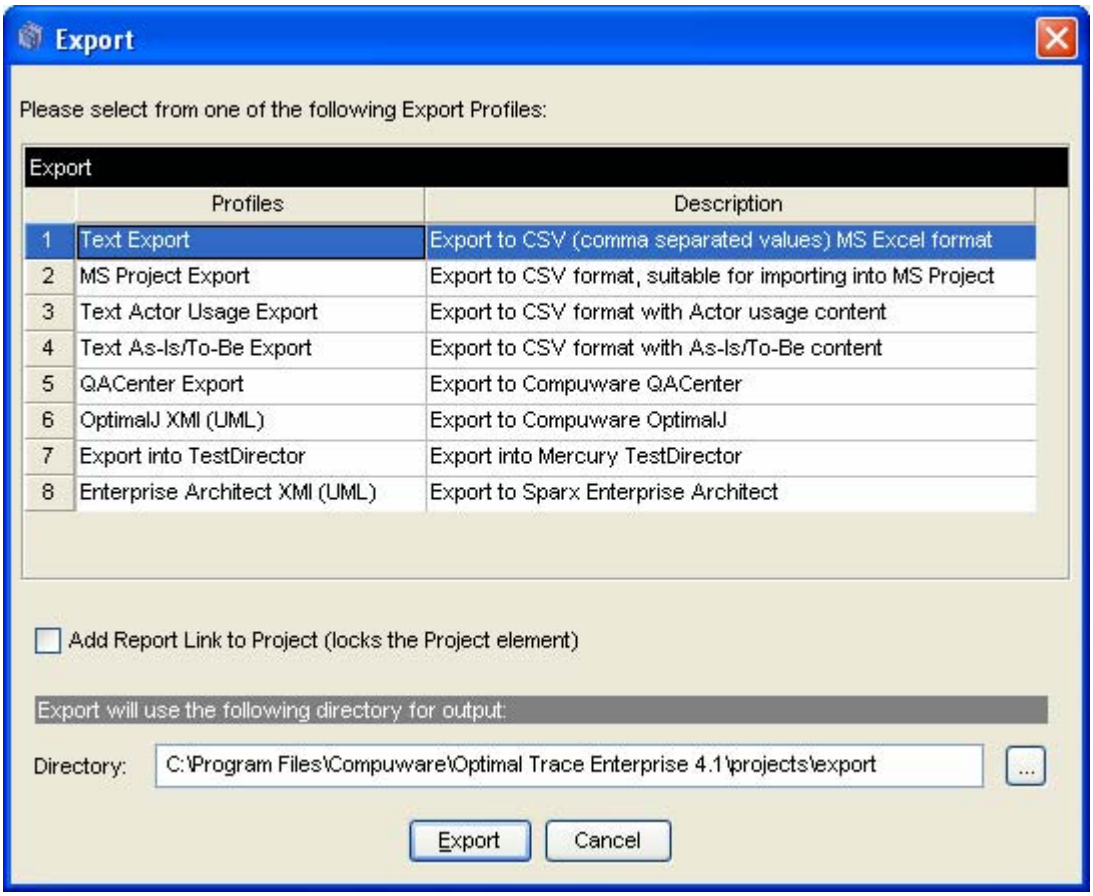
The attributes 'IsReadOnly', 'IsLocked' and 'OutputFileName' are legacy entries and not relevant to our export so we leave them as is. Additionally the 'OutputDirectory' attribute is left blank, meaning the output will default to the export directory under the Optimal Trace installation.

In our example, we only have one Template file, 'Requirement-List.txt, so we just need one <TextGenTemplate> node in our XML file.

We need to change the following attributes in our <TextGenTemplate> node:

- Line e. set "OutputFileName" = "Requirement-List.txt"
- Line h. set "ContextObject" = "Project"
- Line i. set "TemplateFileName" = "Requirement-List.txt".
- Line j. set "ContextVariableName" = "project"

So we've now finished configuring our new text generation template. Now, launch Optimal Trace, and bring up the Export dialog (via the 'Project\Export Project…' menu option), and our new Profile should appear:



Clicking on 'Export' should create a file called 'Requirement-List.txt' in the export directory of Optimal Trace.

6 Creating Reports

Similar to Exports,Optimal Trace uses a generic mechanism for finding new Reports. On start up, it will search the user's personal htmlgen directory and if it sees a suitable Report Profile, it will dynamically populate the Report List in the tool.

By adding to this folder a directory containing your Report files and starting Optimal Trace, the new Report will automatically be added to the list.

When Optimal Trace generates a report it uses the profile.xml template to determine what Optimal Trace Element types are needed for the output (specifically the ContextObject and ContextVariableName variables).

For each element type that it finds (Project, UseCase, Scenario etc.) it will apply the template to *every instance of that Element type in the project.* i.e. if the profile.xml contains references

Note on the Optimal Trace API:
Optimal Trace speaks in terms of Requirements, with each Requirement comprised of a set of scenarios and steps. From the API perspective, each Requirement is termed a 'UseCase'. This stems from early versions of Optimal Trace that spoke in terms of use cases. With later releases of Optimal Trace this notion has been expanded to include other non-use case specific aspects.

(in the ContextObject and ContextVariableName variables) to Project, UseCase and Step, then for each UseCase a report page will be generated and for each Step a report page will be generated. (There is only one Project to apply the Project template to.)

Create a new directory in your personal HTMLgen directory. For clarity, it is recommended that you use the same name for this directory as for the report you intend to create (this is not mandatory, however).

The easiest way to create a new Optimal Trace Report is to copy an existing report and modify it.

Therefore, out of the pre-supplied reports choose the one closest to what you would like to do, then copy the contents of its directory into the directory you just created. Edit the profile.xml file in the same manner as laid out for exports above, renaming the entries as you require (Name, ExternalLinkName, TemplateFileName and OutputFileName). Also please note, if the report you intend to create is to generate different pages for different Requirement types, you will need a template for each Requirement within the profile.xml file. (As shown in the default_html profile.) Next open the index.HTML file in an editor and modify it as you require.

Note: If you edit the profile.xml file you must restart Optimal Trace for the changes to take affect, but changes can be made to the *.HTML file(s) and implemented (by creating the relevant report) while Optimal Trace is running.

The rest of this section will focus on creating a report from scratch (it will also be of use to those modifying a report).

A blank profile.xml profile is available in section 10 Appendix 1: Blank Profile.xml.
A blank index.HTML profile is available in 11 Appendix 2: Blank html Starter..
These can be altered and then saved into the desired directory to create a new report.

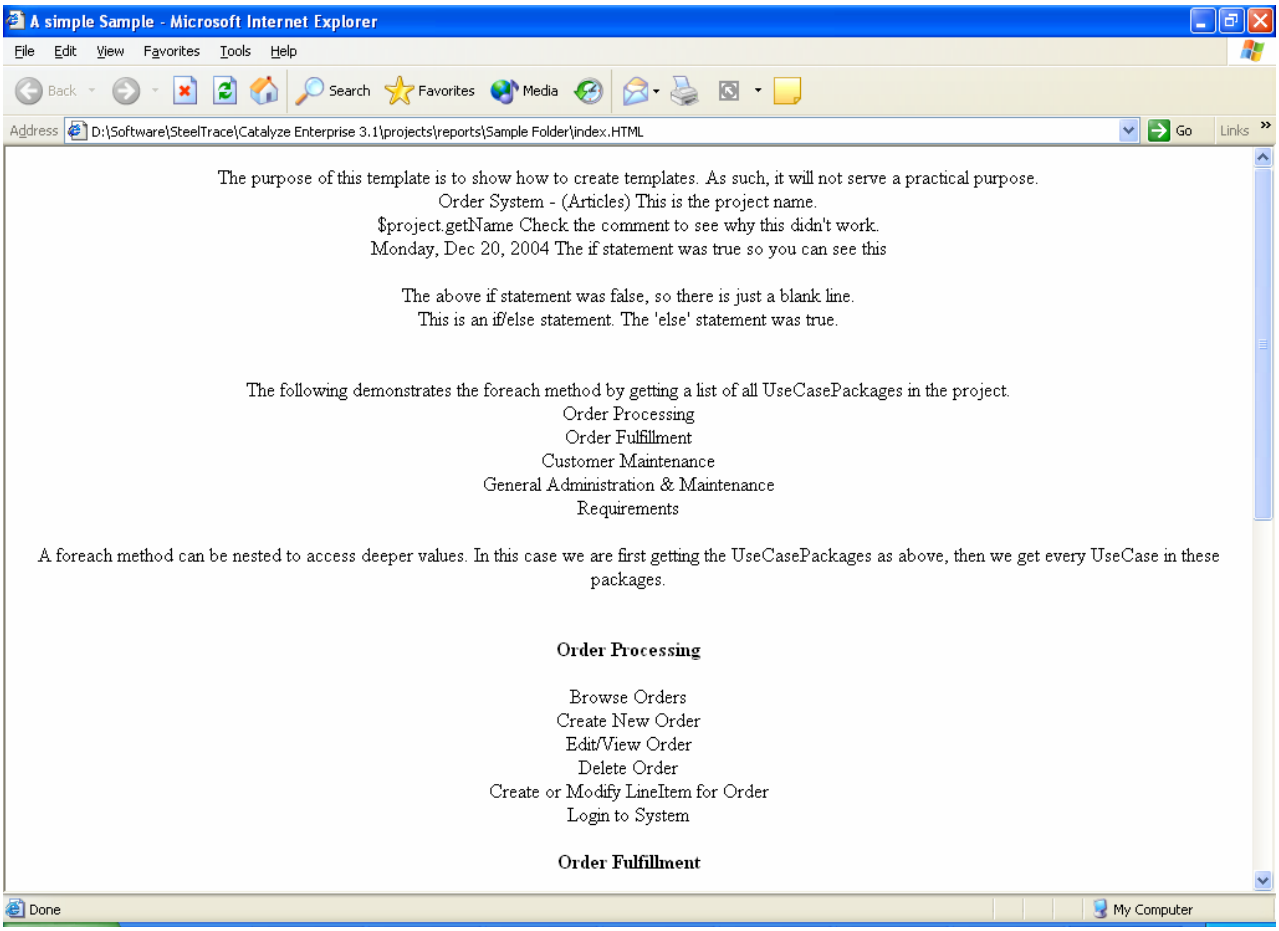
6.1 Example 1: A Simple Report.

This report itemizes the full content of a Optimal Trace project, iterating through all packages and in turn all requirements in this packages.

This report uses 'get*' API commands, 'if' and 'if/else' statements and 'foreach' loops. A full list of these commands can be found in section 9 The Optimal Trace API.

This report provides a basic introduction to the workings of Velocity & Reports.

The result of generating this report appears below.



6.1.1 The Report Profile

The profile.xml file is shown below. Comments embedded in the file outline the intent of each entry.

```
<TextGenProfile Id="ST282810241458958" TimeStamp="1014065346295">
<DynAttributes>
<DynAttribute Name="Position" TimeStamp="1014065346295" Type="java.lang.Integer" Value="7"/>
<!--This will place it at position 7 in the list of available reports. (If it is free) -->
<DynAttribute Name="Description" TimeStamp="1014065346275" Type="java.lang.String" Value="An introductory sample report."/>
```

```
6.  <!--The description of the project that will appear in Optimal Trace -->
7.  <DynAttribute Name="Name" TimeStamp="1014065346275" Type="java.lang.String" Value="Example Report 1"/>
8.  <!--The name that will appear in Optimal Trace.-->
9.  <DynAttribute Name="isReadOnly" TimeStamp="1014065346275" Type="java.lang.Boolean" Value="false"/>
10.<DynAttribute Name="isLocked" TimeStamp="1014065346275" Type="java.lang.Boolean" Value="false"/>
11.<DynAttribute Name="OutputDirectory" TimeStamp="1014065346295" Type="java.lang.String"
     Value="projects\reports\Example Report 1"/>
12.<!--The folder that the generated report will be placed in. -->
13.<DynAttribute Name="OutputFileName" TimeStamp="1014065346295" Type="java.lang.String" Value="index.HTML"/>
14.<!--The name of the file that will be generated. -->
15.<DynAttribute Name="ExternalLinkName" TimeStamp="1014065346295" Type="java.lang.String" Value="Simple
     Sample - HTML"/>
16.<!--A name for linking.-->
17.</DynAttributes>
18.<TextGenTemplate Id="ST282810400007383" TimeStamp="1014065346285">
19.<DynAttributes>
        a.  <DynAttribute Name="Name" TimeStamp="1014065346285" Type="java.lang.String" Value="Index"/>
        b.  <!--The name. -->
        c.  <DynAttribute Name="IsActivityDiagramNeeded" TimeStamp="1014065346285" Type="java.lang.Boolean"
            Value="false"/>
        d.  <DynAttribute Name="IsUseCaseDiagramNeeded" TimeStamp="1014065346285"
            Type="java.lang.Boolean" Value="false"/>
        e.  <DynAttribute Name="Description" TimeStamp="1014065346285" Type="java.lang.String" Value="An
            introductory sample report."/>
        f. <!--Description. -->
        g.  <DynAttribute Name="OutputFileName" TimeStamp="1014065346285" Type="java.lang.String"
            Value="index.HTML"/>
        h.  <!--The name of the file that will be created. -->
        i. <DynAttribute Name="AdditionalFilesToCopy" TimeStamp="1014065346285" Type="java.lang.String"
            Value=""/>
        j. <DynAttribute Name="isLocked" TimeStamp="1014065346285" Type="java.lang.Boolean" Value="false"/>
        k.  <DynAttribute Name="ContextObject" TimeStamp="1014065346285" Type="java.lang.String"
            Value="Project"/>
        l. <DynAttribute Name="TemplateFileName" TimeStamp="1014065346285" Type="java.lang.String"
            Value="HTMLgen\Example Report 1\index.HTML"/>
        m.  <!-- Where to find the template for this profile.-->
        n.  <DynAttribute Name="ContextVariableName" TimeStamp="1014065346285" Type="java.lang.String"
            Value="project"/>
        o.  <DynAttribute Name="isReadOnly" TimeStamp="1014065346285" Type="java.lang.Boolean"
            Value="false"/>
20.</DynAttributes>
21.</TextGenTemplate>
22.</TextGenProfile>
```

Lets look at some sections of the profile. The first section from Line 2 thru Line 17 set the report up in terms of how the report will be interpreted and used from Optimal Trace. We consider these in more detail:

Line 5:

This shows the description of the report as it shows in Optimal Trace.

Line 7:

This is the name of the report as it shows in Optimal Trace.

Line 11:

The output directory that the report will be generated to.

Line 13:

The output file name that is required in Optimal Trace to open the generated reported after generation. Note that this entry should be consistent with the first TextGenTemplate entry for the file name.

Line 15:

This controls the Name of the Link that is inserted into Optimal Trace.

Line 18 thru 21:

This section represents directives relating to the output of the main page.

Save this profile.xml in the 'htmlgen \Example Report 1' directory.

This will display a profile called 'Example Report 1' with a description of 'An Introductory Sample Report' at the seventh position in the Reports Generation dialogue. It will base the template off "index.HTML" which it will look for in the 'htmlgen\Example Report 1' directory.

6.1.2 The Velocity Script(s)

The index.HTML file controls what the report will contain. Below is the script file. Save this as index.HTML in the 'htmlgen\Example Report 1' directory.

```
1. #macro (escapeChars $str)
2. #parse ("escapeChars.vm")
3. #end
4. <HTML>
5. <head>
6. <title>A simple Sample</title>
7. <meta http-equiv="Content-Type" content="text/HTML; charset=iso-8859-1">
8. <link rel="stylesheet" href="Optimal Trace.css" type="text/css">
9. </head>
10.<body bgcolor="#FFFFFF" text="#000000">
11.#*What is displayed here is a blend of HTML and velocity script. The HTML functions as normal, while velocity is used
      to
12.access core Optimal Trace objects. Please check the API for details on these objects. (The hash-star and star-hash
      are used to
13.mark comments in velocity.)*#
14.<div align="center">
15.The purpose of this template is to show how to create templates. As such, it will not serve a practical purpose.
16.##The above is HTML and is visible. This is a velocity comment and is not.
17.<br>
18.##To get the project name do this:
19.$project.getName() This is the project name.
20.#*Any method that returns something will display it. getName returns a String, then velocity puts that String
21.in the document as HTML.*#
22.<br>
23.$project.getName Check the comment to see why this didn't work.
24.#*If velocity doesn't recognise something it will just display it as is. In this case, it doesn't know that getName
25.is a method and doesn't recognise it as a variable.*#
26.<br>
27.##An if statement
28.#if($project.getAllUseCasePackages().size()>0)
29.$project.getCreationDateAsString() The if statement was true so you can see this
30.#end ##Always close if statements.
31.<br>
32.#if($project.getAllUseCasePackages().size()<0)
33.$project.getCreationDateAsString()
34.#end ##Always close if statements.
35.<br>
```

```
36.The above if statement was false, so there is just a blank line.
37.<br>
38.#if($project.getAllUseCasePackages().size()<0)
39.$project.getCreationDateAsString()
40.#else
41.This is an if/else statement. The 'else' statement was true.
42.#end

43.<br><br><br>
44.The following demonstrates the foreach method by getting a list of all UseCasePackages in the project.

45.<br>
46.#foreach($element in $project.getAllUseCasePackages())
47.$element
48.<br>
49.#end
50.<br>
51.A foreach method can be nested to access deeper values. In this case we are first getting the UseCasePackages as
      above, then we get every UseCase in these packages.
52.<br>
53.<br>
54.#foreach($element in $project.getAllUseCasePackages())
55.<br>
56.<h4>$element</h4>##Show which package it is with html header h4.
57.#foreach($element1 in $element.getAllUseCasesInPackages())
58.$element1##Show the UseCase.
59.<br>
60.#end
61.#end
62.<br>
63.</body>
64. </HTML>
```

What is displayed here is a blend of HTML and velocity script. The HTML functions as normal, while velocity is used to access core Optimal Trace objects. Please check the API for details on these objects.

Line 1 thru 3:

This is a macro that escapes any characters in a given string.

Line 4 thru 10:

This is standard HTML and is visible when the report is generated.

Line 11 thru 13:

Hash-star (#*) and star-hash (*#) are used to open and close comments in Velocity. ## is also used to mark single line comments.

Line 19:

This will get and display the project name.

Any method that returns something will display it. getName() returns a String, then velocity puts that String in the document as HTML.

Line 23:

If velocity doesn't recognise a method call and cannot resolve it, it simply displays the text 'as is'. In this case the syntax is wrong (it should be used as ‘getName()’). As it is represented it expects a variable called 'getName' which does not resolve.

Line 28 thru 30:

This is an ‘if’ statement. If it evaluates as true it will get and display the creation date of the project.

Line 31 thru 61:

The remainder of the report demonstrates a mixture of velocity and html. It's primary aim is to list all Packages and the Requirements (names of) contained therein.

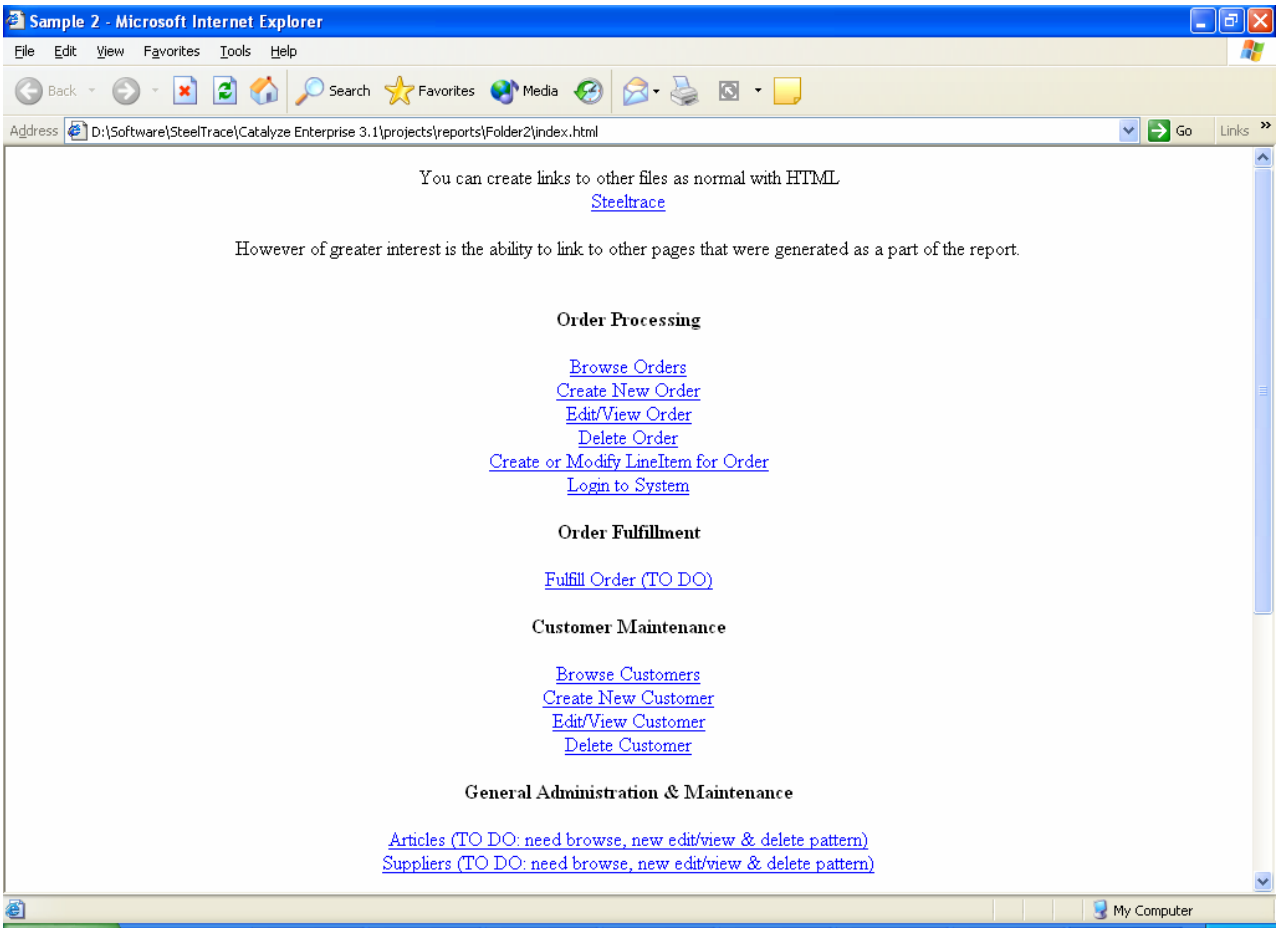
6.2 Example 2: A multi-page Report.

Somewhat similar to the previous report, this report generates a page with the high level contents of the project. In this case however, the report also has a hyperlink to the Optimal Trace website followed by each Structured Requirement in the project sorted by package. Each Requirement is linked (using HTML hrefs) to the detail of the element which links to the specific detail.

Whereas the first example outputted a single HTML page, this example outputs both the single 'master' page and one page (containing detail) for each itemized structured requirement within of the project. This covers each package and each Requirement.

This is achieved by having an additional section in the profile.xml file. This third section (detailed below) references 'UseCase' elements, so that when the report is generated all such elements have a separate page generated to match the requirements of the profile.xml file.

Once you have created the directory & files, run Optimal Trace, open a project and generate a report as normal to see this report.



6.2.1 The Report Profile

The profile.xml file is shown below. Comments embedded in the file outline the intent of each entry.

This report profile generates two file types, the basic "index" file which is responsible for generating the 'master' page and the Requirements (UseCases) that the index links to. As such, it has two TextGenTemplate sections.

```
1. <TextGenProfile Id="ST282810241458958" TimeStamp="1014065346295">
2. <DynAttributes>
3. <DynAttribute Name="Position" TimeStamp="1014065346295" Type="java.lang.Integer" Value="10"/>
4. <DynAttribute Name="Description" TimeStamp="1014065346275" Type="java.lang.String" Value="A more advanced
     sample report."/>
5. <DynAttribute Name="Name" TimeStamp="1014065346275" Type="java.lang.String" Value="Example Report 2"/>
6. <DynAttribute Name="isReadOnly" TimeStamp="1014065346275" Type="java.lang.Boolean" Value="false"/>
7. <DynAttribute Name="isLocked" TimeStamp="1014065346275" Type="java.lang.Boolean" Value="false"/>
8. <DynAttribute Name="OutputDirectory" TimeStamp="1014065346295" Type="java.lang.String"
     Value="projects\reports\Example Report 2"/>
9. <DynAttribute Name="OutputFileName" TimeStamp="1014065346295" Type="java.lang.String" Value="index.html"/>
10.<DynAttribute Name="ExternalLinkName" TimeStamp="1014065346295" Type="java.lang.String" Value="Links
     Sample - HTML"/>
11.</DynAttributes>
```

```
12.<TextGenTemplate Id="ST282810400007383" TimeStamp="1014065346285">
13.<DynAttributes>
        a.  <DynAttribute Name="Name" TimeStamp="1014065346285" Type="java.lang.String" Value="Index"/>
        b.  <DynAttribute Name="IsActivityDiagramNeeded" TimeStamp="1014065346285" Type="java.lang.Boolean"
            Value="false"/>
        c.  <DynAttribute Name="IsUseCaseDiagramNeeded" TimeStamp="1014065346285"
            Type="java.lang.Boolean" Value="false"/>
        d.  <DynAttribute Name="Description" TimeStamp="1014065346285" Type="java.lang.String" Value="A more
            advanced sample report."/>
        e.  <DynAttribute Name="OutputFileName" TimeStamp="1014065346285" Type="java.lang.String"
            Value="index.html"/>
        f. <DynAttribute Name="AdditionalFilesToCopy" TimeStamp="1014065346285" Type="java.lang.String"
            Value=""/>
        g.  <DynAttribute Name="isLocked" TimeStamp="1014065346285" Type="java.lang.Boolean" Value="false"/>
        h.  <DynAttribute Name="ContextObject" TimeStamp="1014065346285" Type="java.lang.String"
            Value="Project"/>
        i. <DynAttribute Name="TemplateFileName" TimeStamp="1014065346285" Type="java.lang.String"
            Value="htmlgen/Example Report 2/index.html"/>
        j. <DynAttribute Name="ContextVariableName" TimeStamp="1014065346285" Type="java.lang.String"
            Value="project"/>
        k.  <DynAttribute Name="isReadOnly" TimeStamp="1014065346285" Type="java.lang.Boolean"
            Value="false"/>
14.</DynAttributes>
15.</TextGenTemplate>
16.<TextGenTemplate Id="ST282810400007233" TimeStamp="1014065346285">
17.<DynAttributes>
        a.  <DynAttribute Name="Name" TimeStamp="1014065346285" Type="java.lang.String" Value="Use Cases"/>
        b.  <DynAttribute Name="IsActivityDiagramNeeded" TimeStamp="1014065346285" Type="java.lang.Boolean"
            Value="false"/>
        c.  <DynAttribute Name="IsUseCaseDiagramNeeded" TimeStamp="1014065346285"
            Type="java.lang.Boolean" Value="false"/>
        d.  <DynAttribute Name="Description" TimeStamp="1014065346285" Type="java.lang.String" Value="Sample
            2-use cases."/>
        e.  <DynAttribute Name="OutputFileName" TimeStamp="1014065346285" Type="java.lang.String"
            Value="UseCase$unique.html"/>
        f. <DynAttribute Name="AdditionalFilesToCopy" TimeStamp="1014065346285" Type="java.lang.String"
            Value=""/>
        g.  <DynAttribute Name="isLocked" TimeStamp="1014065346285" Type="java.lang.Boolean" Value="false"/>
        h.  <DynAttribute Name="ContextObject" TimeStamp="1014065346285" Type="java.lang.String"
            Value="UseCase"/>
        i. <DynAttribute Name="TemplateFileName" TimeStamp="1014065346285" Type="java.lang.String"
            Value="htmlgen/Example Report 2/usecase.html"/>
        j. <DynAttribute Name="ContextVariableName" TimeStamp="1014065346285" Type="java.lang.String"
            Value="usecase"/>
        k.  <DynAttribute Name="isReadOnly" TimeStamp="1014065346285" Type="java.lang.Boolean"
            Value="false"/>
18.</DynAttributes>
19.</TextGenTemplate>
20. </TextGenProfile>
```

Lets look at some sections of the profile.

Line 1 thru 11:

This is the normal set of attributes that specify where the outputs etc. are to be generated to.

Line 12 thru 15:

This represents the directives for the resulting index.htm 'master' page.

Line 16 thru 19:

This section represents directives relating to the output of individual detail pages linked from the master page. As mentioned previously this section is used to dictate how individual . This is the first time we have seen this structure

Two lines should be noted that are new in terms of what we have looked at thus far and it is of some use to explain these further.

Line 17e is leveraging a reserved constant called $unique. The purpose of this is to guarantee that the generated individual files have a unique file name. Hence the result of this example is to generate a number of output files that have as prefix 'usecase' followed by a unique id followed by the suffix '.html'. The net effect of this is that the all output files will be unique. The best and most extensive example of this approach is the default report where many links are created to packages, steps etc in addition to the details associated with the individual requirements. In thie case of the default report there is one 'TextGen Template' required for each file type. In other words one entry in the profile for Step, Package etc.

Line 17h should also be highlighted. This provides the context for the specific velocity script that will be used. You can think of 'context' as being equivalent to the scope. Hence we have an ability to reference in the corresponding Template file the specific scope set in this line.

Save this profile.xml in the 'htmlgen\Example Report 2' directory.

This will display a profile called 'Example Report 2' with a description of 'A more advanced sample report' at the eight position in the Reports Generation dialogue. It will base the template off "index.HTML" which it will look for in the 'htmlgen\ Example Report 2' directory.

6.2.2 The Velocity Script(s)

The index.HTML file controls what the 'master' page in the report will display.

Just as we had two logical sections in the profile.xml file, we have two velocity script files, one each for the 'index' and 'usecase'.

Below are the script files. Save these respectively as index.HTML & usecase.HTML in the 'htmlgen\Example Report 2' directory.

Here is the first; 'index.html'

```
#macro (escapeChars $str)
#parse ("escapeChars.vm")
#end
<HTML>
<head>
<title>Sample 2</title>
<meta http-equiv="Content-Type" content="text/HTML; charset=iso-8859-1">
<link rel="stylesheet" href="Optimal Trace.css" type="text/css">
</head>

<body bgcolor="#FFFFFF" text="#000000">
<div align="center">

You can create links to other files as normal with HTML
<br>
<a href="http://www.Optimal Trace.com">Optimal Trace</a>
<br>
<br>
However of greater interest is the ability to link to other pages that were generated as a part of the report.
```

```
18.<br>
19.<br>
20.##Get all UseCasePackages in the project, and cycle through them.
21.#foreach($element in $project.getAllUseCasePackages())
22.<br><h4>$element</h4>
23.##Get all UseCases in each UseCasePackage
24.#foreach($element1 in $element.getAllUseCasesInPackages() )
25.#*Create a link to a file of name: UseCase+IdNumber. The profile.xml file causes all UseCase files to be named as
      UseCase+IdNumber, thus making each link unique*#
26.<a href="UseCase${element1.getId()}.HTML">$element1</a>
27.<br>
28.#end
29.#end
30.<br>
31.<br>

32.</body>
33. </html>
```

Line 26 is of specific interest and leverages the previously mentioned reserved constant called $unique. Since the id will be unique in the file name, we can use this as the href argument. This results in the HTML link resolving to the appropriately generated filename.

The remainder of the script is standard HTML.

Now lets look at the 2nd script file.

```
1.  #macro (escapeChars $str)
2.  #parse ("escapeChars.vm")
3.  #end
4.  <HTML>
5.  <head>
6.  <title>Not quite so simple sample</title>
7.  <meta http-equiv="Content-Type" content="text/HTML; charset=iso-8859-1">
8.  <link rel="stylesheet" href="Optimal Trace.css" type="text/css">
9.  </head>

10.<body bgcolor="#FFFFFF" text="#000000">
11.<div align="center">
12.$usecase.getDisplayName()

13.<br>
14.<br>
15.</body>
16.</HTML>
```

This is a very simple example of a detail output accompanying the link. For a comprehensive example of this approach refer to the General Report' that ships with Optimal Trace.

Note, that since our context or scope is the 'use case' we have the ability to query directly on this.

Line 12 is the only velocity call while the remainder is HTML. The method call 'getDisplayName' resolves to the named requirement plus the number prefix (correlating to the package tag and number) as shown in the Tree in Optimal Trace. The method 'getName' will not include the prefix. For example 'SR2:Browse Orders' would be output from the above code versus 'Browse Orders' if the 'getName' call is used. See the full API section for full details.

6.3 Example 3: Displaying Increments of Delivery - filtering on Custom Properties

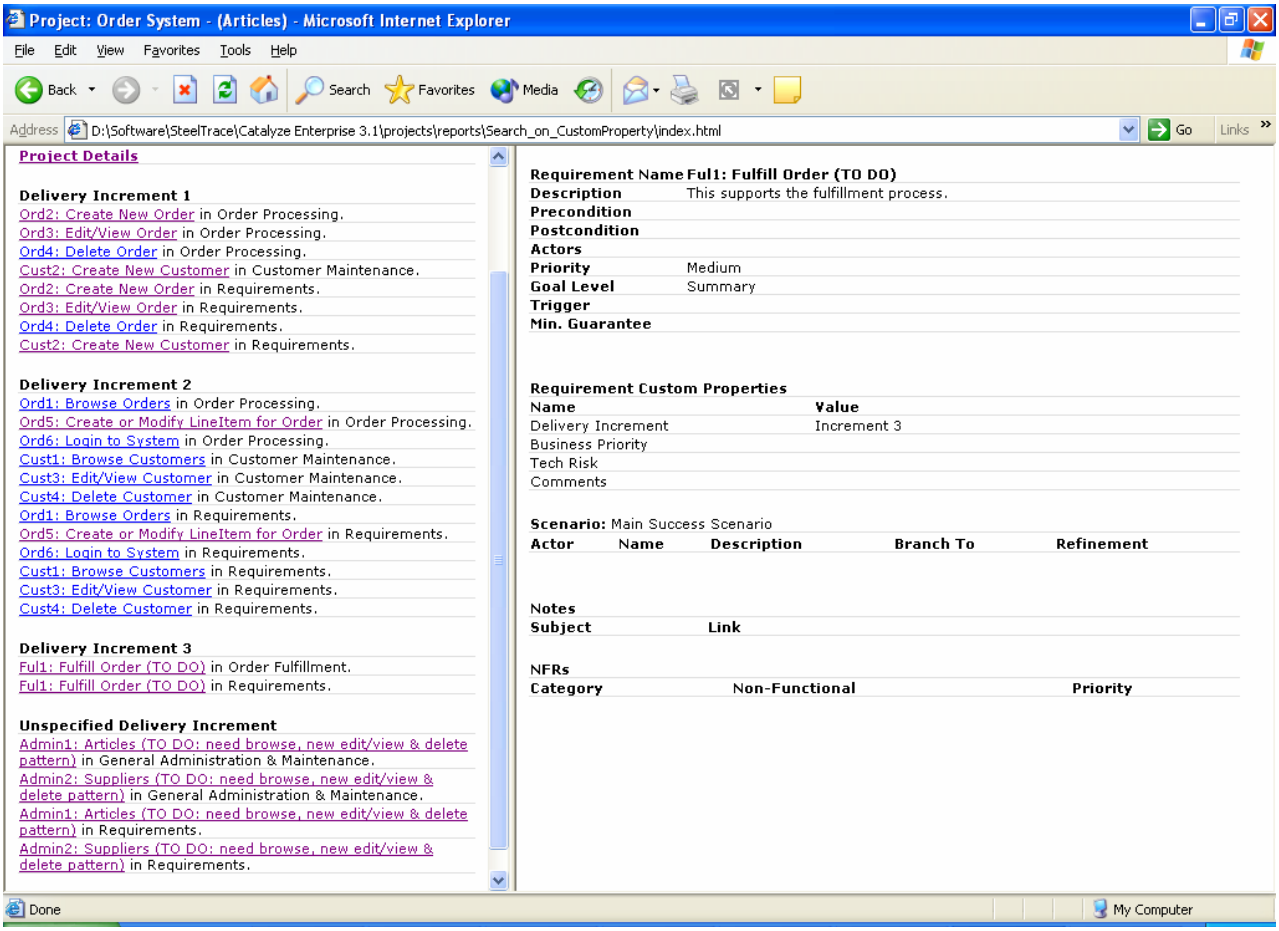
This report demonstrates how you can filter a given project by Custom Properties. This is an extremely useful way of slicing the project in given ways and publishing to stakeholders.

You could easily adapt this report to query on custom fields like 'Risk', 'Business Priority' etc.

In this instance the report in question is customized to expect a custom property called 'Delivery Increment'. This property is contained by default in the 'Standard Software Development' Project Template shipping with Optimal Trace. Our intent is to easily identify what is contained in which increments of delivery.

If you have not created your project originally from that template, add a custom field at the requirement level to run this.

Specifically, this report cycles through each requirement and add it to the list in the left hand frame according to its delivery increment bound values of: 'Increment 1', 'Increment 2', 'Increment 3' or 'Unspecified'. A sample output for this report can be seen below.



6.3.1 The Report Profile

The profile.xml file is shown below. Comments embedded in the file outline the intent of each entry.

In this example, since we have a frames based approach in the generated report there are many file types generated, in effect one per detail section that is required each having a TextGenTemplate section specified.

```
<TextGenProfile Id="ST282810241458958" TimeStamp="1014065346295">
<DynAttributes>
<DynAttribute Name="Position" TimeStamp="1014065346295" Type="java.lang.Integer" Value="10"/>
<DynAttribute Name="Description" TimeStamp="1014065346275" Type="java.lang.String" Value="Sort by custom property."/>
<DynAttribute Name="Name" TimeStamp="1014065346275" Type="java.lang.String" Value="Example Report 3"/>
<DynAttribute Name="isReadOnly" TimeStamp="1014065346275" Type="java.lang.Boolean" Value="false"/>
<DynAttribute Name="isLocked" TimeStamp="1014065346275" Type="java.lang.Boolean" Value="false"/>
<DynAttribute Name="OutputDirectory" TimeStamp="1014065346295" Type="java.lang.String" Value="projects\reports\Example Report 3"/>
<DynAttribute Name="OutputFileName" TimeStamp="1014065346295" Type="java.lang.String" Value="index.html"/>
<DynAttribute Name="ExternalLinkName" TimeStamp="1014065346295" Type="java.lang.String" Value="Links Sample - HTML"/>
</DynAttributes>
<TextGenTemplate Id="ST282810400007383" TimeStamp="1014065346285">
<DynAttributes>
    a. <DynAttribute Name="Name" TimeStamp="1014065346285" Type="java.lang.String" Value="Index"/>
    b. <DynAttribute Name="IsActivityDiagramNeeded" TimeStamp="1014065346285" Type="java.lang.Boolean" Value="false"/>
    c. <DynAttribute Name="IsUseCaseDiagramNeeded" TimeStamp="1014065346285" Type="java.lang.Boolean" Value="false"/>
    d. <DynAttribute Name="Description" TimeStamp="1014065346285" Type="java.lang.String" Value="UseCases by CustomProperty"/>
    e. <DynAttribute Name="OutputFileName" TimeStamp="1014065346285" Type="java.lang.String" Value="index.html"/>
    f. <DynAttribute Name="AdditionalFilesToCopy" TimeStamp="1014065346285" Type="java.lang.String" Value="htmlgen/Optimal Trace.css, htmlgen/Optimal Tracelogo.gif"/>
    g. <DynAttribute Name="isLocked" TimeStamp="1014065346285" Type="java.lang.Boolean" Value="false"/>
    h. <DynAttribute Name="ContextObject" TimeStamp="1014065346285" Type="java.lang.String" Value="Project"/>
    i. <DynAttribute Name="TemplateFileName" TimeStamp="1014065346285" Type="java.lang.String" Value="htmlgen/Example Report 3/index.html"/>
    j. <DynAttribute Name="ContextVariableName" TimeStamp="1014065346285" Type="java.lang.String" Value="project"/>
    k. <DynAttribute Name="isReadOnly" TimeStamp="1014065346285" Type="java.lang.Boolean" Value="false"/>
</DynAttributes>
</TextGenTemplate>
<TextGenTemplate Id="ST282810564953195" TimeStamp="1014065346295">
<DynAttributes>
    a. <DynAttribute Name="Name" TimeStamp="1014065346295" Type="java.lang.String" Value="Index Frame"/>
    b. <DynAttribute Name="IsActivityDiagramNeeded" TimeStamp="1014065346295" Type="java.lang.Boolean" Value="false"/>
    c. <DynAttribute Name="IsUseCaseDiagramNeeded" TimeStamp="1014065346295" Type="java.lang.Boolean" Value="false"/>
    d. <DynAttribute Name="Description" TimeStamp="1014065346295" Type="java.lang.String" Value="Template for Index Frame"/>
    e. <DynAttribute Name="OutputFileName" TimeStamp="1014065346295" Type="java.lang.String" Value="IndexFrame.html"/>
    f. <DynAttribute Name="isLocked" TimeStamp="1014065346295" Type="java.lang.Boolean" Value="false"/>
    g. <DynAttribute Name="ContextObject" TimeStamp="1014065346295" Type="java.lang.String" Value="Project"/>
    h. <DynAttribute Name="TemplateFileName" TimeStamp="1014065346295" Type="java.lang.String" Value="htmlgen/Example Report 3/IndexFrame.html"/>
    i. <DynAttribute Name="ContextVariableName" TimeStamp="1014065346295" Type="java.lang.String" Value="project"/>
    j. <DynAttribute Name="isReadOnly" TimeStamp="1014065346295" Type="java.lang.Boolean" Value="false"/>
</DynAttributes>
```

```
19.</TextGenTemplate>
20.<TextGenTemplate Id="ST282810568134765" TimeStamp="1014065346295">
21.<DynAttributes>
        a. <DynAttribute Name="Name" TimeStamp="1014065346295" Type="java.lang.String" Value="Project Detail
           Frame"/>
        b. <DynAttribute Name="IsActivityDiagramNeeded" TimeStamp="1014065346295" Type="java.lang.Boolean"
           Value="false"/>
        c. <DynAttribute Name="IsUseCaseDiagramNeeded" TimeStamp="1014065346295"
           Type="java.lang.Boolean" Value="false"/>
        d. <DynAttribute Name="Description" TimeStamp="1014065346295" Type="java.lang.String"
           Value="Template for Project Detail Frame"/>
        e. <DynAttribute Name="OutputFileName" TimeStamp="1014065346295" Type="java.lang.String"
           Value="ProjectDetail.html"/>
        f. <DynAttribute Name="isLocked" TimeStamp="1014065346295" Type="java.lang.Boolean" Value="false"/>
        g. <DynAttribute Name="ContextObject" TimeStamp="1014065346295" Type="java.lang.String"
           Value="Project"/>
        h. <DynAttribute Name="TemplateFileName" TimeStamp="1014065346295" Type="java.lang.String"
           Value="htmlgen/Example Report 3/projectdetail.html"/>
        i. <DynAttribute Name="ContextVariableName" TimeStamp="1014065346295" Type="java.lang.String"
           Value="project"/>
        j. <DynAttribute Name="isReadOnly" TimeStamp="1014065346295" Type="java.lang.Boolean" Value="false"/>
22.</DynAttributes>
23.</TextGenTemplate>
24.<TextGenTemplate Id="ST282810400007233" TimeStamp="1014065346285">
25.<DynAttributes>
        a. <DynAttribute Name="Name" TimeStamp="1014065346285" Type="java.lang.String" Value="Use Cases"/>
        b. <DynAttribute Name="IsActivityDiagramNeeded" TimeStamp="1014065346285" Type="java.lang.Boolean"
           Value="false"/>
        c. <DynAttribute Name="IsUseCaseDiagramNeeded" TimeStamp="1014065346285"
           Type="java.lang.Boolean" Value="false"/>
        d. <DynAttribute Name="Description" TimeStamp="1014065346285" Type="java.lang.String" Value="Sample
           UseCases."/>
        e. <DynAttribute Name="OutputFileName" TimeStamp="1014065346285" Type="java.lang.String"
           Value="UseCase$unique.html"/>
        f. <DynAttribute Name="AdditionalFilesToCopy" TimeStamp="1014065346285" Type="java.lang.String"
           Value=""/>
        g. <DynAttribute Name="isLocked" TimeStamp="1014065346285" Type="java.lang.Boolean" Value="false"/>
        h. <DynAttribute Name="ContextObject" TimeStamp="1014065346285" Type="java.lang.String"
           Value="UseCase"/>
        i. <DynAttribute Name="TemplateFileName" TimeStamp="1014065346285" Type="java.lang.String"
           Value="htmlgen/Example Report 3/usecase.html"/>
        j. <DynAttribute Name="ContextVariableName" TimeStamp="1014065346285" Type="java.lang.String"
           Value="usecase"/>
        k. <DynAttribute Name="isReadOnly" TimeStamp="1014065346285" Type="java.lang.Boolean"
           Value="false"/>
26.</DynAttributes>
27.</TextGenTemplate>
28.</TextGenProfile >
```

Lets look at some sections of the profile.

Line 12 thru 15:

This section specifies the index file that sets up the frames.

Line 13f:

This line specifies additional files to copy into the destination report.

Line 13h:

This line specifies the context or scope of the report in this case it is the Project itself.

Line 13i:

The index.html script file is specified as being the relevant script file to be used.

Line 13j:

The variable name that will be used to access the context in the script file.

Line 16 thru 19:

This section controls the output in the left frame of the report. The specific field entries are similar to the previous section with the context and context variable being the same. Here however the script file is different as the HTML and velocity will be different

Line 20 thru 23:

This section controls the output in the right frame that is displayed after selecting the project link in the left frame.

The specific field entries are similar to the previous section with the context and context variable being the same. In this case however, the primary purpose is to display the details at the project only level.

Line 24 thru 27:

This section controls the output in the right frame that is displayed after selecting the a given requirement link in the left frame.

The specific field entries are different in this case since our context is at the specific requirement (use case) level. The purpose is to display the full details associated with the clicked entity that appears in the left frame.

6.3.2 The Velocity Scripts

Since this is a frames based report there are a number of logical sections of script, one for each type of display area. In other words, just as we had a number of logical sections in the profile.xml file we now have a number of script files.

These are:

- Index.html
- IndexFrame.html
- ProjectDetail.html
- UseCase.html

Below are the respective script files.

The Index.html.

```
1. <html>
2. <head>
3. <title>Project: $project.Name</title>
4. <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
5. </head>
6. ##Create the frames
7. <frameset cols="40%,60%" rows="*">
8. ##The left hand frame holds the Index
```

```
9.  <frame name="projectFrame" src="IndexFrame.html">
10.##The right hand frame defaults to the details of the project.
11.<frame name="detailFrame" src="ProjectDetail.html">
12.</frameset>
13. </html>
```

This is fully HTML with no velocity and the purpose is to house both the Index area as well as the detailed frames.

Line 9 refers to the left side of the report while **Line 11** denotes the right side of the report.

The IndexFrame.html follows:

```
1.  #macro (escapeChars $str)
2.  #parse ("escapeChars.vm")
3.  #end
4.  <html>
5.  <head>
6.  <title>Project: #escapeChars($project.Name)</title>
7.  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
8.  ##The Stylesheet is used to give a uniform appearance to the report.
9.  <link rel="stylesheet" href="Optimal Trace.css" type="text/css">
10.</head>
11.<body bgcolor="#FFFFFF" text="#000000">
12.<FONT SIZE=-2>
13.<p><a href="http://www.Optimal Trace.com" target="_blank"><img src="Optimal Tracelogo.gif" width="162"
      height="72" border="0"></a></p>
14.##Tables are used for formatting.
15.<table width="100%" rows="3" border="0" cellpadding = "2">
16.<tr>
17.<td>
18.<table width="100%"  border="0" cellspacing="1" class="containment-border">
19.<tr>
20.<td width="100%" ><b><a href="projectdetail.html" target="detailFrame">Project Details</a></b></td>
21.</tr>
22.<tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr>
23.<tr>
24.<td width="100%" ><b>Delivery Increment 1</b></td>
25.</tr>
26.##Cycle through each package in the project.
27.#foreach($element in $project.getAllUseCasePackages())
28.##Cycle through each requirement in this package
29.#foreach($usecase in $element.getAllUseCasesInPackages())
30.##If the custom property matches our requirements
31.#if($usecase.getCustomProperty('Delivery Increment').getValue()=='Increment 1')
32.<tr>
33.##Insert a link to that requirement.
34.<td width="100%" ><a href="UseCase${usecase.getId()}.html" target="detailFrame">$usecase.getDisplayName()</a>
      in $element.</td>
35.</tr>
36.#end
37.#end
38.#end
39.<tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr>
40.<tr>
41.<td width="100%" ><b>Delivery Increment 2</b></td>
42.</tr>
43.##As above but for a different custom property value
44.#foreach($element in $project.getAllUseCasePackages())
45.#foreach($usecase in $element.getAllUseCasesInPackages())
46.#if($usecase.getCustomProperty('Delivery Increment').getValue()=='Increment 2')
47.<tr>
48.<td width="100%" ><a href="UseCase${usecase.getId()}.html" target="detailFrame">$usecase.getDisplayName()</a>
      in $element.</td>
49.</tr>
50.#end
```

```
#end
#end
<tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr>
<tr>
<td width="100%" ><b>Delivery Increment 3</b></td>
</tr>
##As above but for a different custom property
#foreach($element in $project.getAllUseCasePackages())
#foreach($usecase in $element.getAllUseCasesInPackages())
#if($usecase.getCustomProperty('Delivery Increment').getValue()=='Increment 3')
<tr>
<td width="100%" ><a href="UseCase${usecase.getId()}.html" target="detailFrame">$usecase.getDisplayName()</a>
    in $element.</td>
</tr>
#end
#end
#end
<tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr>
<tr>
<td width="100%" ><b>Unspecified Delivery Increment</b></td>
</tr>
##As above but for a different custom property value
#foreach($element in $project.getAllUseCasePackages())
#foreach($usecase in $element.getAllUseCasesInPackages())
#if($usecase.getCustomProperty('Delivery Increment').getValue()=='')
<tr>
<td width="100%" ><a href="UseCase${usecase.getId()}.html" target="detailFrame">$usecase.getDisplayName()</a>
    in $element.</td>
</tr>
#end
#end
#end
<tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr>
</table>
</table>
</font>
</body>
</html>
```

The ProjectDetail.html follows:

```
#macro (escapeChars $str)
#parse ("escapeChars.vm")
#end
<html>
<head>
<title>Project: #escapeChars($project.Name)</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
##The Stylesheet is used to give a uniform appearance to the report.
<link rel="stylesheet" href="Optimal Trace.css" type="text/css">
</head>
<body bgcolor="#FFFFFF" text="#000000">
<FONT SIZE=-2>
<p><a href="http://www.Optimal Trace.com" target="_blank"><img src="Optimal Tracelogo.gif" width="162"
    height="72" border="0"></a></p>
##Tables are used for formatting.
<table width="100%" rows="3" border="0" cellpadding = "2">
<tr>
<td>
<table width="100%"  border="0" cellspacing="1" class="containment-border">
<tr>
<td width="100%" ><b><a href="projectdetail.html" target="detailFrame">Project Details</a></b></td>
</tr>
<tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr>
```

```
<tr>
<td width="100%" ><b>Delivery Increment 1</b></td>
</tr>
##Cycle through each package in the project.
#foreach($element in $project.getAllUseCasePackages())
##Cycle through each requirement in this package
#foreach($usecase in $element.getAllUseCasesInPackages())
##If the custom property matches our requirements
#if($usecase.getCustomProperty('Delivery Increment').getValue()=='Increment 1')
<tr>
##Insert a link to that requirement.
<td width="100%" ><a href="UseCase${usecase.getId()}.html" target="detailFrame">$usecase.getDisplayName()</a>
    in $element.</td>
</tr>
#end
#end
#end
<tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr>
<tr>
<td width="100%" ><b>Delivery Increment 2</b></td>
</tr>
##As above but for a different custom property value
#foreach($element in $project.getAllUseCasePackages())
#foreach($usecase in $element.getAllUseCasesInPackages())
#if($usecase.getCustomProperty('Delivery Increment').getValue()=='Increment 2')
<tr>
<td width="100%" ><a href="UseCase${usecase.getId()}.html" target="detailFrame">$usecase.getDisplayName()</a>
    in $element.</td>
</tr>
#end
#end
#end
<tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr>
<tr>
<td width="100%" ><b>Delivery Increment 3</b></td>
</tr>
##As above but for a different custom property value
#foreach($element in $project.getAllUseCasePackages())
#foreach($usecase in $element.getAllUseCasesInPackages())
#if($usecase.getCustomProperty('Delivery Increment').getValue()=='Increment 3')
<tr>
<td width="100%" ><a href="UseCase${usecase.getId()}.html" target="detailFrame">$usecase.getDisplayName()</a>
    in $element.</td>
</tr>
#end
#end
#end
<tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr>
<tr>
<td width="100%" ><b>Unspecified Delivery Increment</b></td>
</tr>
#foreach($element in $project.getAllUseCasePackages())
#foreach($usecase in $element.getAllUseCasesInPackages())
#if($usecase.getCustomProperty('Delivery Increment').getValue()!='Increment 1' &&
    $usecase.getCustomProperty('Delivery Increment').getValue()!='Increment 2' &&
    $usecase.getCustomProperty('Delivery Increment').getValue()!='Increment 3' )
<tr>
<td width="100%" ><a href="UseCase${usecase.getId()}.html" target="detailFrame">$usecase.getDisplayName()</a>
    in $element.</td>
</tr>
#end
#end
#end
<tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr><tr></tr>
</table>
```

```
</table>
</font>
</body>
</html>
```

and finally here is the 'UseCase.html':

```
#macro (escapeChars $str)
#parse ("escapeChars.vm")
#end
<html>
<head>
<title>Project: #escapeChars($project.Name)</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link rel="stylesheet" href="Optimal Trace.css" type="text/css">
</head>

<body bgcolor="#FFFFFF" text="#000000">
<div align="center">
<table width="100%"  border="0" class="containment-border" cellspacing="1">
<tr>
<td width="22%"><b>${velocitySupport.getReadableName("Use Case")} Name</b></td>
<td width="78%"><b>#escapeChars($usecase.DisplayName)</b></td>
</tr>
<tr>
<td width="22%"><b>Description</b></td>
<td width="78%">#escapeChars($usecase.BusinessDescription)</td>
</tr>
<tr>
<td width="22%"><b>Precondition</b></td>
<td width="78%">#escapeChars($usecase.PreCondition)</td>
</tr>
<tr>
<td width="22%"><b>Postcondition</b></td>
<td width="78%">#escapeChars($usecase.PostCondition)</td>
</tr>
<tr>
<td width="22%"><b>${velocitySupport.getReadableName("Actors")}</b></td>
<td width="78%">#escapeChars($usecase.Actors)</td>
</tr>
<tr>
<td width="22%"><b>Priority</b></td>
<td width="78%">#escapeChars($usecase.Priority)</td>
</tr>
<tr>
<td width="22%"><b>Goal Level</b></td>
<td width="78%">#escapeChars($usecase.GoalLevel.Name)</td>
</tr>
<tr>
<td width="22%"><b>Trigger</b></td>
<td width="78%">#escapeChars($usecase.Trigger)</td>
</tr>
<tr>
<td width="22%"><b>Min. Guarantee</b></td>
<td width="78%">#escapeChars($usecase.MinimalGuarantee)</td>
</tr>
</table>
<br>

#if ($usecase.getCustomPropertyBucket())
#if ($usecase.getCustomPropertyBucket().getCustomProperties().size() >0)
<br>
<table width="100%"  border="0" cellspacing="1" class="containment-border">
```

```
<tr>
<td colspan="3" ><b>${velocitySupport.getReadableName("Use Case")} Custom Properties</b></td>
</tr>
<tr>
<td width="40%" ><b>Name</b></td>
<td width="60%" ><b>Value</b></td>
</tr>
#foreach ($cp in $usecase.getCustomPropertyBucket().getCustomProperties())
<tr>
<td width="40%" >#escapeChars($cp.Name)</td>
<td width="60%" >#escapeChars($cp.Value)</td>
</tr>
#end
</table>
#end
#end

<br>
#set($dot = ".")
#foreach ($scenario in $usecase.Scenarios)
<table cellpadding="2" width="100%" class="containment-border" border="0" cellspacing="1">
<tr>
<td width="20%"><b>${velocitySupport.getReadableName("Scenario")}:
    </b>#escapeChars($scenario.getName())</td>
</tr>
</table>
<table cellpadding="2" width="100%" class="containment-border" border="0" cellspacing="1">
<th align="left">${velocitySupport.getReadableName("Actor")}</th>
<th align="left">Name</th>
<th align="left">Description</th>
<th align="left">${velocitySupport.getReadableName("Branch")} To</th>
<th align="left">${velocitySupport.getReadableName("Refinement")}</th>
#if ($scenario.getSteps().size() >0)
#set ($firstStep = $scenario.getSteps().get(0))
#if ($firstStep.getCustomPropertyBucket())
#if ($firstStep.getCustomPropertyBucket().getCustomProperties().size() >0)
#foreach ($cp in $firstStep.getCustomPropertyBucket().getCustomProperties())
<th align="left">#escapeChars($cp.Name)</th>
#end
#end
#end
#end
#foreach ($step in $scenario.Steps)
<tr>
#if ($step.Actor)
<td>#escapeChars($step.getActor().getName())</td>
#else
 <td></td>
 #end
 <td>$step.Name</td>
 <td>#escapeChars($step.Description)</td>
 <td>#escapeChars($step.getBranchesAsString(", "))</td>
 <td>#escapeChars($step.getRefinementsAsString(", "))</td>
 #if ($step.getCustomPropertyBucket())
 #if ($step.getCustomPropertyBucket().getCustomProperties().size() >0)
 #foreach ($cp in $step.getCustomPropertyBucket().getCustomProperties())
 <td align="left">#escapeChars($cp.Value)</td>
 #end
 #end
 #end
 </tr>
 #end
 </table>
 <br>
 #end
```

```
<br>

<table width="100%" border="0" class="containment-border" cellspacing="1">
<tr>
<td colspan="2"><b>${velocitySupport.getReadableName("NoteBucket")}</b></td>
</tr>
<tr>
<td width="25%"><b>Subject</b></td>
<td width="75%"><b>${velocitySupport.getReadableName("External Link")}</b></td>
</tr>

#foreach ($note in $usecase.Notes)
<tr>
<td width="25%">#escapeChars($note.Subject)</td>
<td width="75%">#escapeChars($note.Name)</td>
</tr>
#end

</table>
<br>

#set ($numCols = 3)
#if ($usecase.getNonFunctionalRequirements().size() > 0)
#set ($n = $usecase.getNonFunctionalRequirements().get(0))
#if ($n.getCustomPropertyBucket())
#set ($numCols = $numCols + $n.getCustomPropertyBucket().getCustomProperties().size())
#end
#end

<table width="100%"  border="0" class="containment-border" cellspacing="1">
<tr>
<td colspan="$numCols" ><b>${velocitySupport.getReadableName("NFRBucket")}</b></td>
</tr>

<tr>
<td><b>Category</b></td>
<td><b>${velocitySupport.getReadableName("Non-Functional Requirement")}</b></td>
<td><b>Priority</b></td>

#if ($usecase.getNonFunctionalRequirements().size() > 0)
#set ($n = $usecase.getNonFunctionalRequirements().get(0))
#if ($n.getCustomPropertyBucket())
#foreach ($cp in $n.getCustomPropertyBucket().getCustomProperties())
<td><b>#escapeChars($cp.getName())</b></td>
#end
#end
#end

</tr>
#foreach ($nfr in $usecase.NonFunctionalRequirements)
<tr>
<td>#escapeChars($nfr.Category)</td>
<td>#escapeChars($nfr.Name)</td>
<td>#escapeChars($nfr.Priority)</td>
#if ($nfr.getCustomPropertyBucket())
#foreach ($cp in $nfr.getCustomPropertyBucket().getCustomProperties())
<td>#escapeChars($cp.getValue())</td>
#end
#end

</tr>
#end
</table>
</div>
</body>
```

```
174. </html>
```

By running the report and seeing what the output yields you should be able to correlate each area of script with the relevant lines.

Of note in this script is how custom properties are enumerated for each Optimal Trace element. Additionally, the use of the call 'getReadableName' in order to resolve 'readable' names of core Optimal Trace elements.

For example, from the Optimal Trace metamodel perspective the type of a structure requirement is codified as a class called 'UseCase' however it's actual readable name is called 'Requirement'

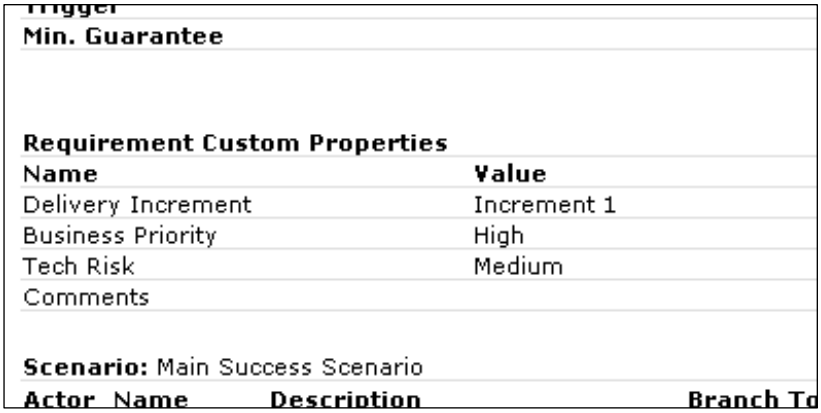
Lets consider some specifics:

Line 14: This is an example of the getReadableName call. In this instance it returns the word 'Requirement' hence the heading text becomes 'Requirement Name'.

Additional examples of this call being made can be found in **Line(s): 30, 56, 76, 80, 83, 84, 121, 125, 144 & 148.**

A full list of legal parameters to this call can be found in section: 9.15 VelocitySupport

Line 51 thru 70: This section of script is responsible for outputting the custom fields for the Structured Requirement (Use Case).



Lines 56 thru 60 sets up the heading area while the remaining lines iterate through each custom property outputting the values of each in turn.

Line 80 thru 94: This section sets up the headings of the steps such that step name, description etc. are output as the first row of a table. Additionally since steps also have custom properties, we iterate through the custom properties to output each name.

Line 95 thru 112: This section now outputs the values for each step including a loop that outputs each custom property value.

Save these in the 'htmlgen\Example Report 3' directory.

7 The Structure of the Profile.xml file

The structure of the profile.xml file is common to both reports and exports. For the purposes of this section, an export profile has been used, but all of this is relevant to the report profile.

Note on Ids & Timestamps in the Profiles:
Optimal Trace profiles contain an Id entry of type long. Although you must ensure the id is present the actual value simply needs to be unique within the scope of the profile.
Timestamp entries are also present. This is legacy and is retained for backward compatibility.
Copying and adjusting existing profile.xml files is the easiest way to start a new profile.

Comments are formed in a *.xml file with the tags <!--to open the comment and --> to close them.

```
<TextGenProfile Id="ST282810241458958" TimeStamp="1014065346295">
```

<TextGenProfile> controls aspects such as how this report will surface in Optimal Trace. (All profile data must be between the < TextGenProfile > and </ TextGenProfile > tags.)

```
<DynAttributes>
<DynAttribute Name="Position" TimeStamp="1014065346295" Type="java.lang.Integer" Value="3"/>
```

This sets position that this export will appear in the list of available exports. If this position is occupied the export will be placed in the closest location available.

```
<DynAttribute Name="Description" TimeStamp="1014065346275" Type="java.lang.String" Value="Export to CSV format with Actor usage content."/>
```

The export description as seen in Optimal Trace

```
<DynAttribute Name="Name" TimeStamp="1014065346275" Type="java.lang.String" Value="Text Actor Usage Export"/>
```

The export name as it appears in Optimal Trace.

```
<DynAttribute Name="isReadOnly" TimeStamp="1014065346275" Type="java.lang.Boolean" Value="false"/>
```

Deprecated - (but required entry)

```
<DynAttribute Name="isLocked" TimeStamp="1014065346275" Type="java.lang.Boolean" Value="false"/>
```

Deprecated - (but required entry)

```
<DynAttribute Name="OutputDirectory" TimeStamp="1014065346295" Type="java.lang.String" Value=""/>
```

This sets the output directory for the export or report. Leaving this blank will output to the default directory: <Optimal Trace Install Directory>\projects\exports. Specifying a folder will (if the folder listed does not exist) create the folder in the <Optimal Trace Installation Directory>\projects\exports folder.

```
<DynAttribute Name="OutputFileName" TimeStamp="1014065346295" Type="java.lang.String" Value="ActorUsage.csv"/>
```

This sets the file name for the generated export (reports are commonly called index.HTML). This is also the name of the file that will be opened when the 'Open' button is clicked after export/report generation.

```
<DynAttribute Name="ExternalLinkName" TimeStamp="1014065346295" Type="java.lang.String" Value="Actor Useage Report - TEXT"/>
```

This sets the name of the link if you check the 'Add Link to Project' option on export/report.

```
</DynAttributes>
```

Close DynAttributes.

```
<TextGenTemplate Id="ST282810400007383" TimeStamp="1014065346285">
```

The <TextGenTemplate> points at the specific velocity template containing the script. This is what will be used to generate the body of the export.

```
<DynAttributes>
```

```
<DynAttribute Name="Name" TimeStamp="1014065346285" Type="java.lang.String"
Value="TextReport"/>
```

This sets the name of the export when it is shown in Optimal Trace.

```
<DynAttribute Name="IsActivityDiagramNeeded" TimeStamp="1014065346285"
Type="java.lang.Boolean" Value="false"/>
```

Reports only & Reserved for future usage

```
<DynAttribute Name="IsUseCaseDiagramNeeded" TimeStamp="1014065346285"
Type="java.lang.Boolean" Value="false"/>
```

This setting is used only for Reports, for exports it should always be set to off. It specifies whether the generation process creates a hot-spotted graphic representing the Requirements Map.

Note: Setting this to 'on' for reports may result in a long duration generation process especially for more complex projects.

```
<DynAttribute Name="Description" TimeStamp="1014065346285" Type="java.lang.String"
Value="CSV file template for Project"/>
```

This gives a description of this file.

```
<DynAttribute Name="OutputFileName" TimeStamp="1014065346285" Type="java.lang.String"
Value="ActorUsage.csv"/>
```

This sets the file name for this template. In the case of a report that contains multiple templates they must all have different names.

```
<DynAttribute Name="AdditionalFilesToCopy" TimeStamp="1014065346285" Type="java.lang.String"
Value=""/>
```

This is only applicable to Reports. In the case of reports, css, graphics and other additional files may be required in the destination directory. Use this setting to specify these files. Company Logos are common examples which are specified when customizing reports.

```
<DynAttribute Name="isLocked" TimeStamp="1014065346285" Type="java.lang.Boolean"
Value="false"/>
```

Deprecated - (but required entry)

```
<DynAttribute Name="ContextObject" TimeStamp="1014065346285" Type="java.lang.String"
Value="Project"/>
```

This specifies the Context (or Object Type) for the Velocity script. The effect of this is that the ContextVariableName below exposes that object type. Only adjust this value for subsidiary outputs. E.g. creating a list of detailed Requirements that will be linked to from the main report page. See Example 3: Displaying Increments of Delivery - filtering on Custom Properties for a comprehensive instance where this setting is fully leveraged.

```
<DynAttribute Name="TemplateFileName" TimeStamp="1014065346285" Type="java.lang.String"
Value="export/Usage/Usage.vm"/>
```

This points to the template to be used with this profile.

```
<DynAttribute Name="ContextVariableName" TimeStamp="1014065346285" Type="java.lang.String"
Value="project"/>
```

This is the initial variable that Velocity script files will use to access the ContextObject. This setting and the 'ContextObject' setting must be synchronized. See Example 3: Displaying Increments of Delivery - filtering on Custom Properties for a comprehensive instance where this setting is fully leveraged.

```
<DynAttribute Name="isReadOnly" TimeStamp="1014065346285" Type="java.lang.Boolean"
Value="false"/>
```

Deprecated - (but required entry)

```
</DynAttributes>
</TextGenTemplate>
</TextGenProfile>
```

Close all open tags.

8 A note on generating MS Word documents with Optimal Trace:

The Velocity text generation mechanism is quite different to the mechanism that Optimal Trace uses to generate MS Word documents. Although similar in concept, in that they use 'Document Profiles', they both use very different architectures internally.

The reason for this is primarily that the MS Word generation is XML based using hidden embedded text in the generated document as it is necessary for users to be able to round-trip Word documents using this mechanism.

The Report and Export mechanism outlined in this whitepaper is ‘one-way’, i.e. a user cannot reverse any content generated.

For more information on developing custom MS Word document profiles, see the Optimal Trace Help documentation.

9 The Optimal Trace API

The Optimal Trace API uses the naming convention of appending 'Ifc' onto each class that represents an interface, so for example ProjectIfc is the name of the Project interface class.

9.1 ProjectIfc

Velocity sample:
```
$project.getName()
#if($project.getAllUseCasePackages().size()>0)
$project.getCreationDateAsString()
#end
```

Type	Name	Description
ActorListIfc	getActorList()	Get the list of Actors participating in this Project.
List	getAllUseCasePackages()	Method to get all UseCasePackageIfc's in a Project.
Long	getCreationDate()	Get the date on which the Project was created.
String	getCreationDateAsString()	Get the date on which the Project was created as a String.
List	getCustomPropertyTemplatesForType()	
CustomPropertyTemplatesMapIfc	getCustomPropertyTemplatesMap()	Get the CustomPropertyTemplatesMapIfc for the Project.
String	getDisplayName()	Get the name to display.
GlossaryIfc	getGlossary()	Get the Glossary associated with this Project.
GoalLevelsIfc	getGoalLevels()	Get the GoalLevels object.
String	getLocation()	Get location of this Project.
String	getLongDescription()	Get the value of scopeGoal
String	getName()	Get the name.
String	getOwner()	Get the owner/creator of this Project.
TableHolderListIfc	getTableHolderList()	Get the TableHolderListIfc for this Project.
String	getTruncatedLocation()	Get the truncated location of this Project.
UseCasePackageIfc	getUseCasePackage()	Get the root UseCase Package. (All other packages and UseCases are children of this Package.)
Long	getVersionLabelCreationDate()	Returns the date at which the version label was applied.
String	getVersionLabelDescription()	Method getVersionLabelDescription
String	getVersionLabelName()	Method getVersionLabelName.
Boolean	isTemplate()	Returns whether or not this is a template.

CustomPropertyIfc	getCustomProperty(String customPropertyName)	Get the CustomPropertyIfc with name customPropertyName. e.g. $project.getCustomProperty('Project Scope').getValue()
CustomPropertyBucketIfc	getCustomPropertyBucket()	Get the Custom Property holder object for this Project.
LinkBucketIfc	getLinkBucket()	Gets the Link holder object for this Project.
NFRBucketIfc	getNFRBucket()	Gets the NFR holder object for this Project.
NoteBucketIfc	getNoteBucket()	Gets the note holder object for this Project.

9.2 ActorListIfc

Type	Name	Description
List	getActors()	Returns the list of ActorIfc objects in the list

9.3 UseCasePackageIfc

Note: for historical reasons, the term 'UseCase' is still in use in the API. In general, a UseCase is 100% equivalent to a Structured Requirement. The exception to this is UseCasePackage, which is a Requirement Package which may contain both Structured Requirements and Simple Requirements.

Velocity Sample:

```
#foreach ($req in $Project.getUseCasePackage.getAllRequirementsPackages())
  $req<br>
#end
```

Type	Name	Description
List	getRequirements()	Returns a list of AbstractRequirementIfc objects which may be structured or simple requirements.
List	getAllRequirementsInPackages()	Returns all the AbstractRequirementIfc objects in this package and all sub-packages.
List	getUseCases()	Get all UseCases (structured requirements) in this package as a list of UseCaseIfc objects.
List	getAllUseCasesInPackages()	Get all UseCases in this package and all sub-packages.
UseCaseIfc	getUseCase(String name)	Get the UseCase named “name”.
List	getSimpleRequirements()	Gets all the SimpleRequirementIfc objects in this package as a list.

List	getAllSimpleRequirementsInPackages()	Gets all the SimpleRequirementIfc objects in this package and all sub-packages.
SimpleRequirementIfc	getSimpleRequirement(String name)	Returns the SimpleRequirementIfc named "name".
List	getUseCasePackages()	Get all UseCasePackages in this package.
List	getAllUseCasePackages()	Get all the UseCasePackages in this Package and all sub-packages.
UseCasePackageIfc	getUseCasePackage(String name)	Get the UseCasePackage named "name".
String	getLongDescription()	Get the description for this UseCasePackage

9.4 AbstractRequirementIfc

An abstract requirement is not a concrete object itself, instead it can be viewed as a base class of both UseCaseIfc (i.e. Structured Requirement) and SimpleRequirementIfc.

Velocity Sample:

```
$ar.getAllAbstractSteps()
$ar.getRefinements()
```

Type	Name	Description
List	getAllAbstractSteps()	Get all the AbstractSteps in this requirement. If a structured requirement, returns the steps in the main scenario and all other scenarios.
String	getBusinessDescription()	Get the BusinessDescription
String	getDisplayName()	Get the AbstractRequirement's display name.

9.5 AbstractStepIfc

An abstract step is the common ancestor of both a step and an item. Concrete instances of this class do not exist, instead they are instances of StepIfc or ItemIfc.

Velocity Sample:

```
$absstep.getDescription()
#if($absstep.getRefinements().size()>0)
$absstep.getRefinementsAsString()
#end
```

Type	Name	Description
String	getDescription()	Get the Step/Item description.
String	getName()	Get the name.
AbstractRequirementIfc	getParentUseCase()	Gets the parent requirement. Note this may be either a simple or structured requirement.
List	getRefinements()	Get a list of RefinementIfcs for this Step/Item.
String	getRefinementsAsString()	Get the Refinements for this Step/Item as a String
Boolean	isCircularRefinement(RefinementIfc Refinement)	This checks for a circular refinement as specified in the Requirements.
Boolean	isRefiningToSelf(AbstractRequirementIfc req)	This checks if a refinement is being made to the requirement that contains this Step/Item.
CustomPropertyIFC	getCustomProperty(java.lang.String, customPropertyName)	Get the CustomPropertyIfc with name customPropertyName. e.g. $step.getCustomProperty('Business Validation').getValue()
CustomPropertyBucketIFC	getCustomPropertyBucket()	Get the Custom Property holder object for this Project.
Boolean	isAlreadyRefinedToUseCase(AbstractRequirementIfc req)	Checks if a refinement to the passed requirement already exists in the Step/Item.

9.6 UseCaseIfc

In addition, UseCaseIfc also inherits all methods in AbstractRequirementIfc.

Velocity Sample:

$usecase.getDisplayName()
$usecase.getBranches()

Type	Name	Description
String	getActors()	Utility method to get all the Actors associated with a UseCase as a String.
List	getAllActors()	Utility method to get all the Actors associated with a UseCase as a List.
List	getAllSteps()	Get all the Steps in all the scenarios for this usecase.
List	getAlternativeScenarios()	Get a list of AlternativeScenarios.

List	getBranches()	Gets the list of Branches pointing to this UseCaseIfc.
List	getBranchesToUseCases()	Gets a list of Branches from Steps in this UseCase that branch to other UseCases.
List	getExtendsUseCases()	Gets a List of UseCases that are extended from this UseCase.
String	getFullyQualifiedName()	Get the fullied qualified name of the UseCase, i.e. considering all its parent packages.
GoalLevelIFC	getGoalLevel()	Get the use case process view level
String	getGraphDisplayOption()	Utility method to return the UseCase's graph display option. Can be one of: SHOW_ALL SHOW_NONE SHOW_MAIN_SUCCESS
List	getIncludesUseCases()	Get a List of UseCases that are included from this UseCase.
MainSuccessScenarioIfc	getMainSuccessScenario()	Get the MainSuccessScenario.
List	getMainSuccessScenarioRefinements()	Get a list of refinements contained in the main success Scenario of this use case.
String	getMinimalGuarantee()	Gets the notes category
String	getName()	Get the name.
Int	getNameLength()	NameLength() utility method primarily for use by Velocity reflection as String.length doesn't work well.
Point2D.Double	getOrigin()	Get the Point2D value of the UseCase's origin
String	getOriginAsString()	Get the Point2D value of the UseCase's origin
Int	getOriginX()	OriginX utility method used by Velocity reflection
Int	getOriginY()	OriginY utility method used by Velocity reflection
String	getPostCondition()	Get the post condition for this Use Case
String	getPreCondition()	Get the pre condition for this Use Case
String	getPriority()	Get the use case priority
List	getRefinements()	Get the refinements, if there are any.
List	getRefinementUseCases()	Get a List of UseCases that are refined from this UseCase.
List	getScenarios()	Get the MainSuccess and Alternate scenarios.
String	getTag()	Get the parent package tag.
String	getTrigger()	Get the trigger for this usecase.
String	getUseCaseNumber()	Get the number associated with the usecase.

UseCasePackageIfc	getUseCasePackage()	Get the use case package that this Use Case belongs to
Boolean	isLeafUsecase()	A leaf use case is a use case that has no more refinements.
Boolean	isRootUseCase	A root use case is a top level use case that is not refined from any other use case
CustomPropertyIFC	getCustomProperty(java.lang.String, customPropertyName)	Get the CustomPropertyIfc with name customPropertyName. e.g. $usecase.getCustomProperty('Technical Risk').getValue()
CustomPropertyBucketIFC	getCustomPropertyBucket()	Get the Custom Property holder object for this Project.
LinkBucketIfc	getLinkBucket()	Gets the Link holder object for this Project.
NFRBucketIfc	getNFRBucket()	Gets the NFR holder object for this Project.
NoteBucketIfc	getNoteBucket	Gets the Note holder object for this Project.
Int	getPositionNumber()	Gets the position number of this Use Case in relation to a RepositionalChildIfc.

9.7 StepIfc

StepIfc also inherits all the methods of AbstractStepIfc. For brevity, these are not repeated here.

Velocity Sample:

```
$step.getDescription()
#if($step.getBranches().size()>0)
$step.getBranchesAsString()
#end
```

Type	Name	Description
ActorIfc	getActor()	Get the Actor associated with this Step
List	getBranches()	Get the branches associated with this Step
String	getBranchesAsString(String separator)	Get the branches associated with this Step as a String
Int	getStepNumber()	Get the Step number.
Boolean	isAlreadyBranched(BizObectIfc bizObject)	This checks if a branch already exists to this business object in a Step.
Boolean	isCircularBranch(BranchIfc branch)	Returns true if the branch points to the Step it came from.

int	getPositionNumber()	Get the Step number.
RefinementIfc	getRefinementToUseCase(UseCaseIfc useCase)	Returns the Refinement that this Step has to a particular use case.
Boolean	isCircularRelationship(UseCaseIfc toUseCase)	Returns true if the Step branches to its parent use case
Boolean	isSiblingRefinement(UseCaseIfc useCase)	Checks for a sibling refinement as specified in the Requirements.

9.8 ScenarioIfc

Velocity Sample:

```
#foreach($scenario in $usecase.getScenarios())
$scenario.getSteps()
#end
```

Type	Name	Description
String	getName()	Get the name.
StepIfc	getStep(Integer stepNumber)	Get Step number stepNumber.
List	getStepRefinements()	Get the list of refinements in the Steps associated with this Scenario
List	getSteps()	Get the list of Steps associated with this Scenario.
Boolean	appearsInFlowDiagram()	Returns whether or not this Scenario appears in the Flow Diagram.

9.9 AlternativeScenarioIfc

Velocity Sample:

```
#foreach($scenario in $usecase.getScenarios())
#if($scenario.getTypeId()=='AlternativeScenario')
$scenario.getDisplayName()
<br>
#end
#end
```

Type	Name	Description
Int	getAlternativeScenarioNumber()	Get the Scenario number.
String	getDisplayName()	Get the Scenario name to display.
String	getDisplayNumber()	Get the Scenario number to display.
String	getName()	Get the name.

Int	getPositionNumber()	Get the Scenario number.

9.10 SimpleRequirementIfc

In addition, SimpleRequirementIfc inherits all the methods of AbstractRequirementIfc.

Velocity Sample:
$simpleReq.getItemList()

Type	Name	Description
ItemListIfc	getItemList()	Returns a list object from which the list of ItemIfc objects can be retrieved.
String	getTagAndNumber()	Returns the tag and simple requirement number for this Simple Requirement as displayed in the Optimal Trace Enterprise application.

9.11 ItemListIfc

The ItemListIfc is a simple list holder which is used simply to retrieve items from a simple requirement.

Velocity Sample:
$iList.getItems()

Type	Name	Description
List	getItems()	Returns a regular list of the ItemIfcs in the item list.

9.12 ItemIfc

ItemIfc also inherits all the methods of AbstractStepIfc.

Velocity Sample:
$item.getName()

Type	Name	Description
String	getDisplayNumber()	Returns the number of the item as displayed in the Optimal Trace Enterprise tool.

9.13 RefinementIfc

Velocity Sample:

```
#foreach($step in $usecase.getAllSteps())
#foreach($refinement in $step.getRefinements())
$refinement.getRefinedToUseCase()
<br>
#end
#end
```

Type	Name	Description
String	getJustification()	Get the Justification for this Refinement
StepIfc	getRefinedFromStep()	Get the Step that this is refined from
UseCaseIfc	getRefinedToUseCase()	Get the UseCase that this refines to
Boolean	isCircularRefinement()	This checks for a circular refinement as specified in the Requirements doc.
Boolean	isFirstCircular()	Get the First circular property for this Refinement
Boolean	isRefiningToSelf()	This checks if a refinement is being made to the use case that contains the refined Step.

9.14 ActorIfc

Velocity Sample:

```
#foreach($step in $usecase.getAllSteps())
$step.getActor().getDefinition()
<br>
#end
```

Type	Name	Description
String	getDefinition()	Get the definition for this ActorIfc
String	getName()	Get the name.
CustomPropertyIFC	getCustomProperty(java.lang.String, customPropertyName)	Get the CustomPropertyIfc with name customPropertyName. e.g. $actor.getCustomProperty('Custom Property').getValue()
CustomPropertyBucketIFC	getCustomPropertyBucket()	Get the Custom Property holder object for this Project.

9.15 VelocitySupport

This is a convenience class that allows normal parsing and other actions that would require substantial velocity coding (or would not be possible in velocity) to be called directly from velocity script. Additionally this provides specific methods such as 'getTraceTree' that traverse the Optimal Trace metamodel returning more complex results not possible in standard velocity.

Velocity Sample:
$velocitySupport.copyAndReverseList($usecase.getAllSteps())
$velocitySupport.getEnumeration($usecase.getAllSteps())

Type	Name	Description
List	copyAndReverseList	This is just a utility method to take a list, make a copy of it and reverse it.
List	copyList(List src)	Just returns a copy of a list. (You cannot instantiate a new list in velocity)
Date	createNewDate()	Returns the current date and time as a java date object.
List	createNewList()	Creates a new empty list object
Map	createNewMap()	Creates a new map
Random	createNewRandom()	Creates a random number
StringTokenizer	createNewStringTokenizer(String String, String delim)	Creates a new java String tokenizer to tokenize the given String with the given delimiter.
String	getOptimal TraceURL(long bizObjectId)	Returns the URL for the object.
Enumeration	getEnumeration(Collection c)	Creates an enumeration from a collection
String	getIconFileNameForLink(String link)	Returns the filename of the icon used for the specific link. Used for HTML reports where a href needs to reference a specific image based on the type of link.
Static VelocitySupport	getInstance()	Returns an instance of the velocity support class.
String	getLinkDisplayName(String link)	Returns the name of the link.
String	getReadableName(String type)	Returns the actual name of a given core Optimal Trace element. The following is a list of valid strings that can be passed as parameters returning the right-hand value: 1. Use Case=Requirement 2. Use Cases=Requirements

3. Actor=Actor
4. Actors=Actors
5. Glossary=Glossary
6. Glossaries=Glossaries
7. DictionaryDefinition=Entry
8. DictionaryDefinitions=Entries
9. Actor List=Actor List
10. Scenario=Scenario
11. Scenarios=Scenarios
12. Alternative Scenario=Alternative Scenario
13. Alternative Scenarios=Alternative Scenario
14. Main Success Scenario=Main Success Scenario
15. Step=Step
16. Steps=Steps
17. Refinement=Refinement
18. Refinements=Refinements
19. Branch=Branch
20. Branches=Branches
21. Non-Functional Requirement=Non-Functional
22. NFRBucket=NFRs
23. External Link=Link
24. LinkBucket=Links
25. Note=Note
26. NoteBucket=Notes

TraceTreeRequirementIfc	getTraceTree(AbstractRequirementIfc req)	Returns the Traceability Tree for a specific requirement.
Boolean	isLinkValid(String link)	For file based links, this returns whether the link can be resolved to a valid filename.
Void	reverseList(List l)	Simply reverses the given list

9.16 TraceTreeRequirementIfc

Int	getNumberOfBranches()	Get the number of Branches, zero for TraceTreeRequirements which have been retrieved from simple requirements.
Int	getNumberOfRefinements()	Get the number of Refinements.
AbstractRequirementIfc	getRequirement()	Returns the associated requirement passed to VelocitySupport.getTraceTree()

9.17 CustomPropertyBucketIfc

Velocity Sample:
$usecase.getCustomPropertyBucket().getCustomProperties()

#end

Type	Name	Description
List	getCustomProperties()	Get a List of the CustomProperties in this CustomPropertyBucket.
CustomPropertyIfc	getCustomPropertyForTemplate(CustomPropertyTemplateIfc template)	Get the Custom Property for the Custom Property Template.
Boolean	hasCustomProperties()	Returns whether or not there are any CustomProperties in this CustomPropertyBucket.

9.18 CustomPropertyIfc

Velocity Sample:
#foreach($customproperty in $usecase.getCustomPropertyBucket().getCustomProperties())
$customproperty.getValue()

#end

Type	Name	Description
List	getAllowedBoundPropertyValues()	Returns the list of allowed values for this custom property.
BoundPropertyValueIfc	getBoundPropertyValue()	This returns what the BoundValue is.
String	getFreeformValue()	
String	getName()	Get the name.
String	getValue()	Get the value of this CustomProperty
String	getValueType()	Get what type of value this Custom Property has. e.g BOUND_VALUE or FREEFORM_VALUE.
Boolean	isBound()	Return whether or not this Custom Property is bound.

9.19 CustomPropertyTemplateIfc

Velocity Sample:
#foreach($template in $project.getCustomPropertyTemplatesForType('UseCase'))
$template.getDefinition()

#end

Type	Name	Description
List	getAllowedBoundPropertyValues()	Get a list of the allowed bound property values.
BoundPropertyValueIfc	getDefaultBoundPropertyValue()	Get the default bound property value.

String	getDefaultFreeformValue()	Get the default Freeform value.
String	getDefaultValue()	Get the default value.
String	getDefinition()	Get the definition.
String	getName()	Get the name.
Boolean	isBound()	Get whether or not this is bound.
BoundPropertyValueBucketIfc	getBoundPropertyValueBucket()	Get the list of bound custom properties

9.20 CustomPropertyTemplatesMapIfc

Velocity Sample:
$CustomPropertyTemplatesMap.getCustomPropertyHolderTypes()
$CustomPropertyTemplatesMap.getCustomPropertyTemplateLists()

Type	Name	Description
Set	getCustomPropertyHolderTypes()	A custom property holder is an object that can contain custom (read only) CustomPropertyHolder properties. These objects are currently UseCase, Step, Package, Project, NFR.
CustomPropertyTemplatesListIfc	getCustomPropertyTemplateListForType(String type)	Get the CustomPropertyTemplateList associated with a Type
List	getCustomPropertyTemplateLists()	Get all the CustomPropertyLists.
String	getTypeIdForTemplateList(CustomPropertyTemplateListIfc child)	Given a descriptive name return the typeid for the type this custom attribute list is used for

9.21 BoundPropertyValueIfc

Type	Name	Description
String	getName()	Get the name.
String	getValue()	Get the value of this BoundPropertyValue.

9.22 BoundPropertyValueBucketIfc

Velocity Sample:
$BoundPropertyValueBucket.getBoundPropertyValues()
$BoundPropertyValueBucket.getTypeId()

Type	Name	Description

List	getBoundPropertyValues()	Get a list of BoundPropertyValues.

9.23 GoalLevelIfc

Velocity Sample:
$usecase.getGoalLevel().getName()

Type	Name	Description
String	getDescription()	Get the description

9.24 GoalLevelsIfc

Velocity Sample:
$project.getGoalLevels().getGoalLevels()

Type	Name	Description
List	getGoalLevels()	Get a list of all GoalLevelIfc's.

9.25 NFRBucketIfc

Velocity Sample:
$usecase.getNFRBucket().getNonFunctionalRequirements()

Type	Name	Description
List	getNonFunctionalRequirements()	Get a list of all NonFunctionalRequirements in this NFRBucket.

9.26 NoteBucketIfc

Velocity Sample:
$project.getNoteBucket().getNotes()

Type	Name	Description
List	getNotes()	Get a list of all notes in this NoteBucket.

9.27 GlossaryIfc

Velocity Sample:
$project.getGlossary().getDictionaryDefs()

Type	Name	Description
List	getDictionaryDefs()	Get a list of all Glossary Items in the glossary (as DictionaryDef objects).

9.28 DictionaryDefIfc

Velocity Sample:

```
#foreach ($Def in $project.getGlossary().getDictionaryDefs())
  $Def.getDefinition()<br>## Displays each definition in the glossary on a new line.
#end
```

Type	Name	Description
String	getReadableName()	Get the name of the Glossaary item
String	getDefinition()	Gets the Glossary Item definition

9.29 BranchIfc

Velocity Sample:

```
#foreach ($Branch in $Step.getBranches())
  $Branch.getCondition()<br>
#end
```

Type	Name	Description
String	getCondition()	Get the Condition for this branch.
Boolean	isOptional()	Return whether this is an optional branch.
StepIfc	getStep()	Get the StepIfc that is the source of this branch.
BizObjectIfc	getBizObjectIfc()	Get the BizObject that is the target of this branch (either a StepIfc, or a UseCaseIfc)

10 Appendix 1: Blank Profile.xml

This is a blank profile.xml selection. Copy and paste this into a text file and save as "profile.xml". Then edit to create the desired template.

```
<TextGenProfile Id="ST282810241458958" TimeStamp="1014065346295">

<DynAttributes>
<DynAttribute Name="Position" TimeStamp="1014065346295" Type="java.lang.Integer" Value=""/>
<DynAttribute Name="Description" TimeStamp="1014065346275" Type="java.lang.String" Value=""/>
<DynAttribute Name="Name" TimeStamp="1014065346275" Type="java.lang.String" Value=""/>
<DynAttribute Name="isReadOnly" TimeStamp="1014065346275" Type="java.lang.Boolean" Value="false"/>
<DynAttribute Name="isLocked" TimeStamp="1014065346275" Type="java.lang.Boolean" Value="false"/>
<DynAttribute Name="OutputDirectory" TimeStamp="1014065346295" Type="java.lang.String" Value=""/>
<DynAttribute Name="OutputFileName" TimeStamp="1014065346295" Type="java.lang.String" Value=""/>
<DynAttribute Name="ExternalLinkName" TimeStamp="1014065346295" Type="java.lang.String"
Value=""/>
</DynAttributes>

<TextGenTemplate Id="ST282810400007383" TimeStamp="1014065346285">

<DynAttributes>
<DynAttribute Name="Name" TimeStamp="1014065346285" Type="java.lang.String" Value=""/>
<DynAttribute Name="IsActivityDiagramNeeded" TimeStamp="1014065346285" Type="java.lang.Boolean"
Value="false"/>
<DynAttribute Name="IsUseCaseDiagramNeeded" TimeStamp="1014065346285" Type="java.lang.Boolean"
Value="false"/>
<DynAttribute Name="Description" TimeStamp="1014065346285" Type="java.lang.String" Value=""/>
<DynAttribute Name="OutputFileName" TimeStamp="1014065346285" Type="java.lang.String" Value=""/>
<DynAttribute Name="AdditionalFilesToCopy" TimeStamp="1014065346285" Type="java.lang.String"
Value=""/>
<DynAttribute Name="isLocked" TimeStamp="1014065346285" Type="java.lang.Boolean" Value="false"/>
<DynAttribute Name="ContextObject" TimeStamp="1014065346285" Type="java.lang.String"
Value="Project"/>
<DynAttribute Name="TemplateFileName" TimeStamp="1014065346285" Type="java.lang.String"
Value="export/Usage/Usage.vm"/>
<DynAttribute Name="ContextVariableName" TimeStamp="1014065346285" Type="java.lang.String"
Value="project"/>
<DynAttribute Name="isReadOnly" TimeStamp="1014065346285" Type="java.lang.Boolean" Value="false"/>
</DynAttributes>

</TextGenTemplate>

</TextGenProfile>
```

11 Appendix 2: Blank html Starter.

This is a selection of HTML that can be used to start a HTML report template from scratch.

```
#macro (escapeChars $str)
#parse ("escapeChars.vm")
```

```
#end
<HTML>
<head>
<title></title>
<meta http-equiv="Content-Type" content="text/HTML; charset=iso-8859-1">
<link rel="stylesheet" href="Optimal Trace.css" type="text/css">
</head>

<body bgcolor="#FFFFFF" text="#000000">


</body>
</HTML>
```

12 Appendix 3: Detailed List of all text exports and reports.

12.1 Report List:

Report Profile	Description
General Report Generates to directory: ...\projects\reports\default_HTML	Generates a frames based Requirements document in HTML format, complete with hyperlinks to all elements of the project.
Swimlane Report Generates to directory: ...\projects\reports\swimlanes_HTML	This report outputs your project with each Scenario displayed with actors/resources on the vertical axis and the steps correlating with actors on the horizontal. This report is extremely useful for identifying steps with no associated resource and resource oriented gap analysis.
Requirement Template (AC) Report Generates to directory: ...\projects\reports\Use-Case-Cockburn-Style_HTML	This report outputs any Optimal Trace project with style and terminology from Alistair Cockburn applied. It is also a very useful report to show a very compact HTML based view of the project.
Actor Usage Report Generates to directory: ...\projects\reports\ResourceUsage_HTML	Generates an actor oriented report with each actor and the steps it participates in. The intent of the report is to show what actors/resources interact with given steps within the Requirements and which steps have no actor currently defined. This is very useful for gap analysis and redundant actor identification.

Traceability Generates to directory: ...\projects\reports\TraceabilityReport_HTML	Generates a report showing all Trace Relationships across the project. The Trace relationships include all refinement, branch and link references. When clicking a given reference the specific element is selected within the Optimal Trace environment.
As Is – To Be Report Generates to directory: ...\projects\reports\As-Is-To-Be-Report_HTML	Generates a report in HTML format with e.g. current details for administrators ('As-Is' processes), and projected details for system developers ('To-Be' processes). The Goal-levels 'As-Is' and 'To-Be' that ship with Optimal Trace are used as discriminators for this report.

12.2 Export List:

Export Profile	Description
Text Export Generates to directory: ...\projects\exports	Export to CSV (Comma-separated) Text File. Generates a report with all the Project details in CSV format. This report is especially good if you wish to output the content of Optimal Trace projects into text editing tools such as notepad or Excel spreadsheets. Since the output is delimited by commas, an excel worksheet can open the output for manipulation.
MS Project Export Generates to directory: ...\projects\exports	Generates a report with all the Project details in CSV format (comma delimited text). Using Optimal Trace supplied Mapping Rules this file may then be opened in MS Project.
Text Actor Usage Export Generates to directory: ...\projects\exports	This report generates CSV based text files (comma delimited text) Generates an actor oriented text output with each actor and the steps it participates in. The intent of the report is to show what actors/resources interact with given steps within the Requirements and which steps have no actor currently defined. This is very useful for gap analysis and redundant actor identification. An HTML version of this export is available under the 'Generation >> Generate Reports...' menu option.

Text As Is – To Be Report Generates to directory: …\projects\exports	This report generates CSV based text files (comma delimited text) Generates a text output with e.g. current details for administrators (‘As-Is’ processes), and projected details for system developers (‘To-Be’ processes). The Goal-levels 'As-Is' and 'To-Be' that ship with Optimal Trace are used as discriminators for this export. An HTML version of this export is available under the 'Generation >> Generate Reports…' menu option.
Optimal J XMI (UML) Generates to directory: …\projects\exports	This report generates an XML file, suitable for use with Compuware Optimal J. The output generated includes: a full listing of Requirements and each object's steps.
Test Director Export Generates to directory: …\projects\exports	This report generates CSV based text files (comma delimited text). The output generated includes: the non-functional Requirements of the project, a full listing of Requirements and each objects steps.