Questa® SIM Verification Management User's Manual

Including Support for Questa SV/AFV

Software Version 10.1b

Table of Contents

List of Examples

List of Figures

List of Tables

Chapter 1
Verification Management Overview

Introduction to Verification Management

Verification Management is a functionality within Questa SIM that offers a wide variety of features for managing your test environment. These features are available in the GUI with the Verification Management Browser and Tracker and are built upon a database called the Unified Coverage DataBase (UCDB). This database has been developed to store all the information necessary to manage the complete Verification flow. The UCDB is used natively within Questa SIM to save all code coverage, functional coverage and assertion data from simulation. See "What is the Unified Coverage Database?" for information on UCDBs.

Note

Documentation for Questa SIM also supports Questa SV/AFV products. For more complete information on current support for Questa SV/AFV, refer to the Installation and Licensing Guide.

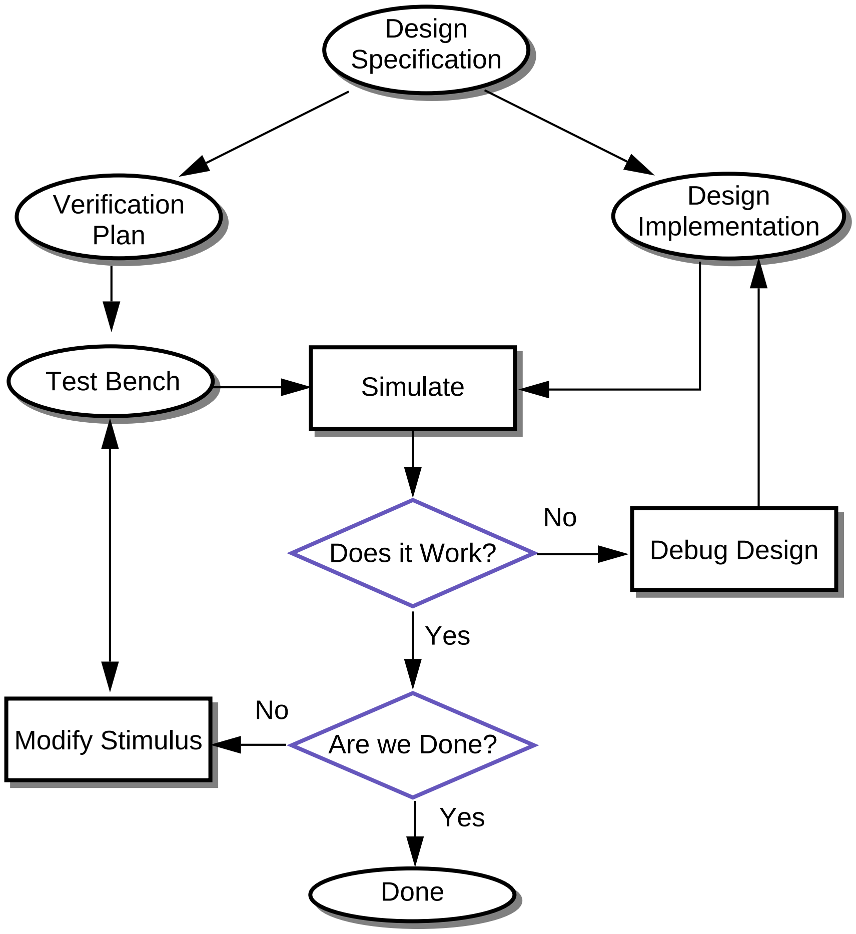
Most of the features of Verification Management are based on the analysis and usage of your verification plan together with collected coverage data. Questa SIM supports the creation of a verification plan in a variety of different word processing and spreadsheet formats. The plan is then imported into a UCDB which is then merged with your test data, so that Questa SIM can display the plan and link it to various coverage items in your design and test bench.

Once a verification plan is imported, you can analyze the traceability of your verification requirements, and measure your overall verification progress against your verification plan. The primary verification management tasks include:

- the merging and aggregation of coverage data
- ranking of tests
- ranking of tests within a testplan
- analysis of coverage in light of late-stage ECO's
- test and command re-runs
- various analyses of coverage data
- generation of easy-to-read HTML coverage reports

The flow described in Figure 1-1 represents a typical design verification process as it can be applied in the Questa SIM environment.

Figure 1-1. Verification of a Design



Every project starts with a design specification. The specification contains elaborate details on the design construction and its intent.

A verification team uses the design specification to create a verification plan. The verification plan contains a list of all questions that need to be answered by the verification process (the golden reference). The verification plan also serves as a functional spec for the test bench.

Once the test bench is built and the designers succeed in implementing the design, you simulate the design to answer the question: “Does it work?”. If the answer is no, the verification engineer gives the design back to designers to debug the design. If yes, it is time to ask the next question: “Are we done yet?”. Answering this question involves investigating how much of the design has been exercised by looking at the coverage data and comparing it against the verification plan.

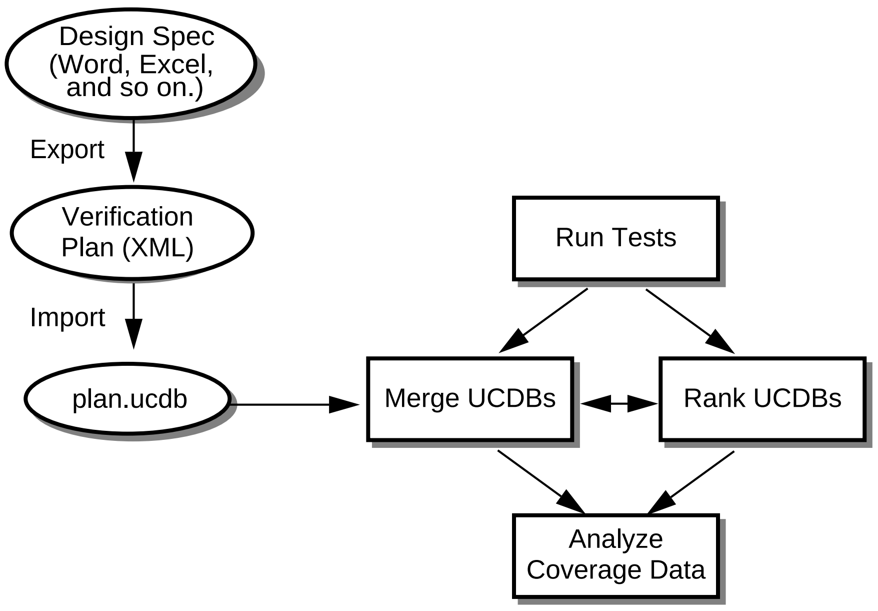
What is Not in this Manual

This manual is intended to guide users through the process of using the various Verification Management tasks within the Questa SIM tool. It is assumed that you already have a basic understanding of, as well as some experience with, the generation of a UCDB, and merging UCDBs. If this is not the case, please see the *Questa SIM SV/AFV User's Manual* for important information relating to the generation of UCDBs, merging, ranking, and the Verification Management Browser.

Verification Management for Analyzing Requirements

Once a verification plan is established, you can analyze the fulfillment of your verification requirements and measure your overall verification progress against the verification plan. An example flow would be as described in Figure 1-2.

Figure 1-2. Verification Management Flow



The flow illustrated above is as follows:

1. Create the verification plan, export it to .xml format, and convert to a UCDB file. See "Exporting a Plan to XML" and "Importing an XML Verification Plan".

   A specified column of the testplan rows associates each section of the testplan with one or more coverage items.

2. Run tests, capturing coverage from each in a UCDB file. See "Collecting and Saving Coverage Data" and "Rerunning Tests and Executing Commands".

3. Merge the verification plan UCDB file and the coverage UCDB files from each run. See "Merging Coverage Data".

The testplan UCDB is annotated by the import utility, and during the merge it links testplan sections with the covergroups. Objects sharing the same tag name are thereby linked together.

4. Rank tests, either before or after merging with the verification plan into a single .ucdb.

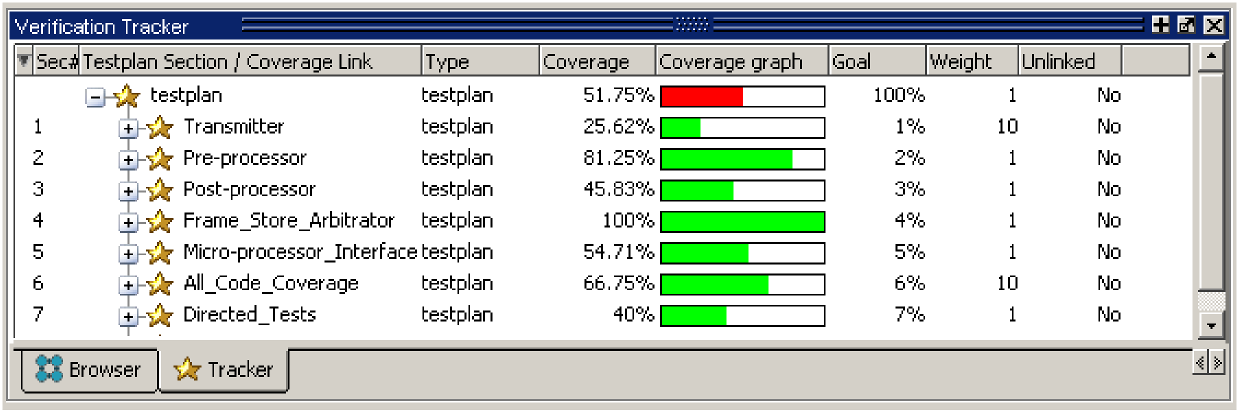
5. Load the UCDB file in Coverage View mode, using vsim -viewcov.

   The coverage analysis command relies on data structures created only when a UCDB file is loaded completely into memory, which only occurs in Coverage View mode (see "Coverage View Mode and the UCDB"). This is why coverage analysis is not available in vsim (simulation) or vcover (the standalone batch utility.) In vsim simulation, UCDB files are only created on disk and never exist in memory; in the vcover batch utility, inputs are never completely loaded into memory and instead use the "read streaming mode" which is much more memory efficient.

6. Analyze the results using the Verification Tracker, Verification Browser, Assertions, Covergroups, and Cover Directives windows, or "coverage analyze" command, to analyze coverage and the testplan.

See "Viewing Test Data in the Tracker Window" for specific instructions on viewing data using the Verification Tracker window. Figure 1-3 shows an imported and merged verification plan, with coverage statistics.

Figure 1-3. Verification Management Test Tracker



Sec#	Testplan Section / Coverage Link	Type	Coverage	Coverage graph	Goal	Weight	Unlinked
	testplan	testplan	51.75%		100%	1	No
1	Transmitter	testplan	25.62%		1%	10	No
2	Pre-processor	testplan	81.25%		2%	1	No
3	Post-processor	testplan	45.83%		3%	1	No
4	Frame_Store_Arbitrator	testplan	100%		4%	1	No
5	Micro-processor_Interface	testplan	54.71%		5%	1	No
6	All_Code_Coverage	testplan	66.75%		6%	10	No
7	Directed_Tests	testplan	40%		7%	1	No

See "Calculation of Total Coverage" for details on how coverage numbers are calculated wherever Total Coverage figures are displayed. Specific sections exist within that section which detail the calculation of numbers in the Verification Tracker and the Verification Browser.

Modes of Use

The UCDB is a format for storing information about the verification environment. The database itself is stored in a persistent form in a binary file. There are three basic methods of accessing the data in the UCDB when using the utilities within Questa SIM:

- Active simulation mode
  (using GUI or vsim command)
- Within post processing or “Coverage View” mode
  (using GUI or “coverage” commands)
- Direct processing of the UCDB file
  (using the “vcover” commands)

The typical usage flow shown in “Introduction to Verification Management” uses a mixture of these modes and will be clearly defined throughout this manual.

XML Terms and Concept Review for Plan Import

To use some of the Questa SIM verification plan and tracking features, it is necessary to have a basic understanding of the XML markup language. This section is provided for those who have little or no experience with XML.

- XML — a way to annotate a plain-text data file with what is called "meta-data". XML is an acronym for "eXtended Markup Language".
- Data — what is being annotated.
- Meta-data — information about the data as opposed to data itself

  For example, consider the boldface type setting in the following phrase:

  this example

  The text "this example" is the data, whereas the information about where the boldface font starts and ends would be the meta-data.
- Markup —When the meta-data information is embedded in the file along with the data, that meta-data information is commonly called "markup".
- Tag — an XML markup identifier. In XML, tags are delineated by a pair of angle brackets.
- Semantic data — Usually, a matched pair of tags surround some bit of data and provide a description of the meaning of the data between those tags.

  For example, in the following entry:

  ```
  <title>My Data</title>
  ```

  the string "My Data" is annotated with a "title" tag pair, possibly denoting that the string is a section title of some sort. The tag name is "title" in this case. The initial "title" tag is called a "start tag". The tag beginning with a slash is called an "end tag" and serves to close the span of text which started at the initial "title" tag.
- Nested tags — tags may be nested, as in the following example:

```
        <title>My <bold>excellent</bold> Data</title>
```

These brief descriptions should be enough to enable you to customize the XML Verification Plan Import for new data formats. For further information, please refer to one of the following documents:

- http://en.wikipedia.org/wiki/XML (an XML overview)
- http://www.w3.org/XML (the XML Specification)

Chapter 2
Capturing and Managing a Verification Plan

The architecture of the UCDB allows it to store sections of a Verification Plan as a tree hierarchy of scopes in a similar way it stores a design hierarchy scopes. Mechanisms within the UCDB format then allow Verification Plan scopes and coverage scopes to be tagged so that an association can be made between them. This allows the plan to be annotated with coverage data and complex queries made to answer questions about the Verification process. It is possible to interface with this infrastructure via the UCDB API and develop very powerful tracking capacities tailored to any users needs.

Questa SIM has utilities that simplify the use of the tagging system and provide a standard product for tracking Verification plans. Verification plans tend to be written using standard editing tools such as Microsoft Word or Excel, Adobe Framemaker or even requirements tracking software such as DOORs from Telelogic. However a plan is captured, there are common pieces of data that will be entered and need to be imported into the UCDB. Within the Questa SIM environment there is an import facility called “xml2ucdb” which is able to read documents in XML and import data into the UCDB.

Tip: Questa SIM offers an easy to use, powerful tool to aid you in creating testplans, linking to coverage objects, and annotating coverage data. See “Questa Excel Add-In for Testplan Creation” for more information.

One of the most important aspects of the Verification plan is the definition of the linking to the coverage objects within the environment. Most coverage objects within the UCDB can be linked to the testplan. The UCDB’s tagging mechanism allows testplan scopes and coverage scopes within the database to be tagged. If a testplan scope and a coverage scope share the same tag, which is a string, then they are associated. This allows coverage to be calculated and queried based upon the testplan.

This chapter includes:

Verification Plan Contents

This section defines and describes the information required in a typical Verification Plan in order to enable many of the powerful capabilities within the Verification Management tool. The sections which follow focus on using Microsoft Word and Excel to capture this information, as well as the method for importing it into a UCDB file.

Columns in the Plan

The columns of data in a testplan are determined positionally. In other words, for an import to work correctly, the columns to be imported must appear in their default (expected) positions in the plan. The positions are set with the “datafields” parameter inside the *xml2ucdb.ini* file (for more information see “Parameter for Mapping by Column Sequence”), or using the -datafields parameter to the xml2ucdb command.

The expected data fields, in order of their expected position within the testplan, are listed in Table 2-1.

Table 2-1. Default Columns in Plan

Data Fields / Columns	Format	Description
Section	<section#>[.<sub-section#>]...	Specifies the section number that is used to determine plan hierarchy, and to generate tags (links to coverage items in the design). “Parent” sections must be defined before “child” sections (i.e. section 1 before 1.1, before 1.1.1, etc.).

Table 2-1. Default Columns in Plan

Data Fields / Columns	Format	Description
Title	<scope_name>	Specifies the title of the "section". The <scope_name> string as entered appears in the VM Tracker window and in the coverage reports. For example, if section 1 has a section title of "Interfaces", and section 1.1 has a section title of "Wishbone", then the hierarchical name of section 1.1 becomes */Interfaces/Wishbone*.
Description	<text>	This is a textual description of the document section. Usually, this would be a description of the testcase or test procedure that the section details. This can be free-form text.
Link	<coverage_item>, <coverage_item>, ...	Specifies what coverage objects or testcases are linked to the testplan section. A testplan section can have one or multiple links to a coverage object or testcase. A specific format is required for the link string, as is a combination of the "Type" and "Path" string for the testplan section.[1] See "Generic Link Entry Syntax Rules" for the link format. See also "Specifying Levels of Hierarchy in Plan".
Type	type of <coverage_item>, type of <coverage_item>, ...	Specifies the type of coverage item referred to in the Link field (see Table 2-7 for a list of valid values and syntax for each Type). These entries are case insensitive. There must be one entry in the "Type" column for each entry found in the "Link" column. However, there can be a single entry in the "Type" column which is applied to all the entries in the "Link" field.[2]
Weight	<weight_as_int>	Specifies an integer value used in the calculation of the weighted averages of the parent sections, calculated in floating point.
Goal	<goal> (1 - 100)	Specifies the percentage value as a goal for a particular coverage object in the GUI. Effectively, this sets the threshold at which the bar-graph goes from uncovered (red) to covered (green) within the Questa SIM GUI. It does not affect the overall coverage grading calculation.
Path	<path_to_linked_item>	An optional data field, not included by default. Specifies the actual instance path to an item in a corresponding Link. Alternatively, this path can precede the string specified in the Link column.

Table 2-1. Default Columns in Plan

Data Fields / Columns	Format	Description
AtLeast	<AtLeast_value>	User must add this column to testplan (see Parameters Mapping Testplan Data Items). Overrides the value of the at_least for functional coverage bins (covergroup and cover directives). Only valid when “Bin” or “Directive” is specified as the Type. For all other types, set value to either “1”, “-”, or “ ” (white space). See Overriding at_least Values in Testplan.
Unimplemented	Yes | No | <integer> Yes — one bin created No — none (default) <integer> — # of bins to be created for this item	User must add this column to testplan (see Parameters Mapping Testplan Data Items). Specifies whether to count the item in the testplan as unimplemented. Any non-zero value affects the coverage calculation for that testplan section. For usage and instructions, see “What are Unimplemented Links and Why Use Them?”.

1. Using this link, type and path data, a command line is calculated for the "coverage tag" command automatically by the xml2ucdb utility. This command is then stored as an attribute of the testplan section to be later used by the merge facility to automatically tag the coverage objects within the simulation UCDB's during the merge. Once a testplan section has linked objects, the coverage number for that testplan section is calculated as the weighted average of the linked objects. Every link requires a corresponding "Type" to be defined so that xml2ucdb can calculate the appropriate tag command for the coverage object.
2. The information in the Link and Type columns is used directly to interpret the command line switches to the "coverage tag" command, which is required for tagging the defined coverage object.

User Defined Attributes

The pre-defined attributes that have been detailed in this section, and listed in Table 2-1 are standard ones within the UCDB. It is also possible to add any user defined attributes to the UCDB. However, the xml2ucdb command that translates the plan from XML to the UCDB format needs to be instructed how to read this extra data. For example, you can add information such as the name of the person responsible for a particular section of the plan, or a priority level for the section of the testplan. The values of these attributes can then be used within the Questa SIM Tracker pane or the command line querying to filter the results based upon user data. Instructions for adding user defined attributes are detailed in “Storing User Attributes in UCDB”. Information on how to use these attributes to filter and query is shown in “Filtering Results by User Attributes”.

Quick Overview to Linking with the Coverage Model

One of the most important aspects of the Verification plan is the definition of the linking to the coverage objects within the environment. Any coverage object within the UCDB can be linked to the testplan. The UCDB has a tagging mechanism that allows testplan scopes and coverage scopes within the database to be tagged. If a testplan scope and a coverage scope share the same tag, which is a string, then they are associated. This allows coverage to be calculated and queried based upon the testplan. The xml2ucdb utility transforms information within the link, type and path values into "coverage tag" syntax, which is then stored as an attribute of the testplan scope. The format used within the link, type and path values is the same regardless of the document format used.

For further details, including an example on using these values to link to the coverage model, see "Links Between the Plan Section and the Coverage Item".

For detailed syntax information for the values used for linking, see "Syntax for Links to the Coverage Model".

Overriding at_least Values in Testplan

You can override at_least values for specific items in your testplan by placing the overriding value into the AtLeast column in the plan UCDB. Whether or not you need to create that column in which to put the overriding value depends on the format in which you write the original UCDB:

- The Word format recognizes an "AtLeast" label placed in the testplan and automatically creates the necessary UCDB column.
- If your testplan originates in Excel or some other format, you need to add "AtLeast" to the *xml2ucdb.ini* configuration file as a column using datafields parameter. For example:

  ```
  datafields = Section,Title,Description,Link,Type,Weight,Goal,AtLeast,Unimplemented
  ```

  The order of the fields is important, and AtLeast must be added at the end. See Parameter for Mapping by Column Sequence for further details on using the datafields parameter.

Related Topics

- Parameters Mapping Testplan Data Items
- Timestamps and UCDB Modification or Merging

Excel Add-In for Creating and Editing Testplans

A Questa Excel Add-In program is available to aid in the creation and editing of testplans. It is by far the easiest method for creating a testplan from scratch. It is distributed along with the Questa SIM installation, located within the install directory. To install the Add-In, run the following:

```
$MTI_HOME/vm_src/QuestaExcelAddIn.msi
```

For complete instructions on installing the program, see *$MTI_HOME/vm_src/HowToInstallExcelAddIn.txt*. For usage information, see "Questa Excel Add-In for Testplan Creation".

Coverage Calculation in Testplans

Note

The coverage calculation algorithms applied to testplans and testplans sections are not the same as those applied to "Total Coverage" calculations, or those applied to calculations of instances and/or design units. See "Calculation of Total Coverage" for details of these aggregation algorithms.

The weight of each testplan section is used to compute the coverage of the parent testplan section according to the "weighted average" method. The weighted average method is the default setting for testplan coverage calculation wherein the coverage grading calculation for a testplan as a whole is the weighted average of its children. This allows portions of the plan to be given more weight than others. For example, a parent testplan section contains two subsections:

- A - contains 99 bins and no bins were hit
- B - contains 1 bin and this bin was hit

Using the weighted average calculation, the coverage calculation for the parent testplan section is 50%.

An additional feature which allows you to specify the relative weights of linked items in one testplan section is the LinkWeight column. See "Excluding Linked Coverage with LinkWeight" for more information.

Also to be considered in coverage calculation is the effect of zero weights within the testplan. When a weight of zero is applied to a testplan section, then this testplan section will not contribute to the coverage value of its parent testplan section.

For examples of the testplan coverage calculation method, see "Examples of Weighted Average Rollup Calculation".

Examples of Weighted Average Rollup Calculation

The coverage calculations are a roll up from the lowest leaf levels up to the top level, taking into consideration the weight values.

Example 2-1. Two sections, both weights = 1

```
|_Top = 37.5%
   |_ Section1 coverage = 25% weight = 1
     |_ Suba coverage = 0% weight = 1
      |_ Subb coverage = 0% weight = 1
```

```
        |_ Subc coverage = 0% weight = 1
        |_ Subd coverage = 100% weight = 1
     |_ Section2 coverage = 50% weight = 1
        |_ Suba coverage = 50% weight = 1
        |_ Subb coverage = 50% weight = 1

  Top = 25 * 1 + 50 * 1  = 37.5%
        ---------------_______
             1 + 1
```

With equal weights for the two sections, the weighted average of Top is 37.5%, as Section 1 is 25% and Section 2 is 50%.

Example 2-2. One section with weight = 2

In this example, we state that section 2 is more important, so the calculation will move closer to the section2 number.

```
  |_Top = 41.66%
     |_ Section1 coverage = 25% weight = 1
        |_ Suba coverage = 0% weight = 1
        |_ Subb coverage = 0% weight = 1
        |_ Subc coverage = 0% weight = 1
        |_ Subd coverage = 100% weight = 1
     |_ Section2 coverage = 50% weight = 2
        |_ Suba coverage = 50% weight = 1
        |_ Subb coverage = 50% weight = 1


  Top = 25 * 1 + 50* 2  = 41.665%
        --------------
             1 + 2
```

So, Top is now 41.665% because the 50% of section 2 is counted twice, and causes the overall calculation to move closer to the higher percentage.

Example 2-3. One section with weight = 0

Now, let's make Section 2 have a weight of 0.

```
  |_Top = 25%
     |_ Section1 coverage = 25% weight = 1
        |_ Suba coverage = 0% weight = 1
        |_ Subb coverage = 0% weight = 1
        |_ Subc coverage = 0% weight = 1
        |_ Subd coverage = 100% weight = 1
     |_ Section2 coverage = 50% weight = 0
        |_ Suba coverage = 50% weight = 1
        |_ Subb coverage = 50% weight = 1

  Top = 25 * 1 + 50 * 0  = 25%
        --------------- ____________
              1 + 0
```

Here, the roll up calculation causes the value of section 2 to be excluded, as it is multiplied by 0 in the numerator, and its weight of zero in the denominator causes it to have no effect at all on the calculation. Thus, the final coverage calculation of Top is 25%.

Guidelines for Writing Verification Plans

This section contains a set of guidelines for you to follow when writing your verification plan to insure that the Verification Plan Import utility properly recognizes the data in your verification plan, written in Excel, Word, or DocBook.

Regardless of the format you use, multiple items (Link, Path, Type, etc.) within an entry in the plan can be separated by one or more whitespace characters (space, tab, newline, or carriage return).

For examples of plans written in Excel, Word and DocBook, and converted from XML to UCDB, see “XML to UCDB Conversion Examples”.

Related Topics

- Excel Add-In for Creating and Editing Testplans
- Parameters Mapping Testplan Data Items
- Importing an XML Verification Plan

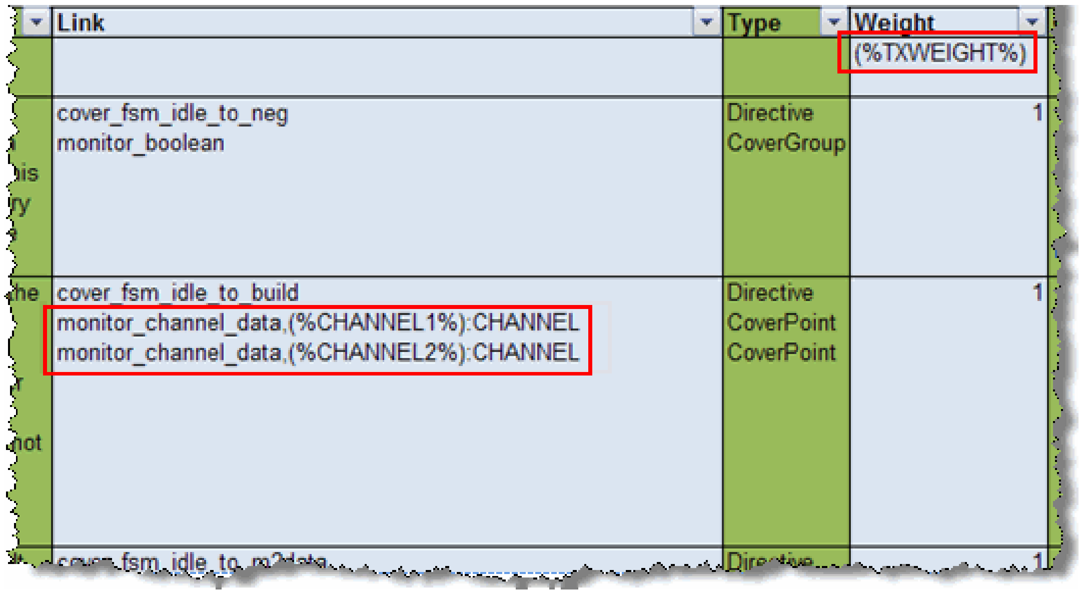
Parameterizing Testplans

You can re-use instance-based testplans. This method of reuse — parameterized testplans — is most useful when dealing with IP, VIP, or for re-use within projects. Parameters can be applied within the testplan source and the values for these can be over-ridden for different uses of the IP, for example testplans for the Mentor Graphics VIPs or Questa Verification IP.

Procedure

1. Place the necessary parameters in the testplan source (Excel, Word, etc.) For example, Figure 2-1 shows an Excel testplan with parameters in the Link and Weight columns.

Figure 2-1. Testplan with Parameters



Link	Type	Weight
		(%TXWEIGHT%)
cover_fsm_idle_to_neg monitor_boolean	Directive CoverGroup	1
cover_fsm_idle_to_build monitor_channel_data,(%CHANNEL1%):CHANNEL monitor_channel_data,(%CHANNEL2%):CHANNEL	Directive CoverPoint CoverPoint	1
cover_fsm_idle_to_m2data	Directive	1

2. Export the testplan to XML. See Exporting a Plan to XML for details.
3. Specify the values for the parameters using any of the following three methods, listed in order of precedence from highest to lowest:
   - Apply -G<var>=<value> arguments to the xml2ucdb command, such as:

     ```
     -GCHANNEL1=chana -GCHANNEL2=chand -GTXWEIGHT=2 -GBINWEIGHT=0
     ```

   - Place the values in a file that is specified by the xml2ucdb command's -varfile argument:

     ```
     xml2ucdb -varfile <file>
     ```

     where <file> contains a list of <variable>=<value> entries, one per line:

     ```
     CHANNEL1=chana
     CHANNEL2=chand
     TXWEIGHT=2
     BINWEIGHT=0
     ```

   - Using the "varfile" setting within the *xml2ucdb.ini* file, such as:

     ```
     varfile=path/myvarfile
     ```

     where *myvarfile* contains the variables and values:

     ```
     CHANNEL1=chana
     CHANNEL2=chand
     TXWEIGHT=2
     BINWEIGHT=0
     ```

4. Import the XML testplan into Questa SIM as instructed in Importing an XML Verification Plan, or the xml2ucdb command.

Results

Once you have completed importing the testplan, you will have unique testplans, each with their own channels and weights.

Related Topics

- Excel Add-In for Creating and Editing Testplans
- xml2ucdb command
- varfile Setting for Parameterizing Testplans
- Parameters Mapping Testplan Data Items
- Importing an XML Verification Plan

Guidelines for Excel Spreadsheets

The columns of data in a testplan are determined positionally. In other words, for an import from an Excel spreadsheet to work correctly, the columns to be imported must appear in their default (expected) positions in the plan. The positions are set in the "datafields" parameter inside the *xml2ucdb.ini* file, or using the -datafields parameter to the xml2ucdb command (see "Parameters Mapping Testplan Data Items"). The expected data fields, in order of their expected position within the testplan, are listed in Table 2-1.

Each row in Excel is interpreted as a testplan section, starting with the first[1] row.

Any rows whose first data field contains only a "#" is interpreted as a comment row, and the contents of that row are not included in the imported UCDB.

Tip: Important: Any blank columns present on the left side of the data must be completely blank (otherwise, the sequence of the columns will be altered).

The PATH datafield is used to specify the path as the starting point for whatever matching operation you request, such as linking to all CoverPoints and CoverGroups recursively.

Guidelines for Word Documents

Microsoft Word supports hierarchical sections using the "Heading" styles.

- Title your top-level testplan sections, using the "Heading1" style.

1. You can customize this starting point by editing the value of the "startstoring" parameter in the xml2ucdb.ini file, the default for which is "1".

- Use the next highest numbered "Heading" style for sub-sections under each section (that is: in section "1.1" under section "1", a "Heading2" style would be used).

  The text of the heading (to which the "Heading" style is attached) becomes the name of the testplan section.

  You can nest your verification plan sections to the extent there are defined "Heading" styles available (up to nine levels in Word 2003).

  When the XML is exported, the sections with a "Heading1" style result in a "wx:section" XML tag and all other sections result in a "wx:sub-section" XML tag. The sections are nested according to their "Heading" style settings like an outline. You do not need to enter section numbers when using this format, because the Verification Plan Importer auto-numbers the sections according to the document hierarchy.

Tip: You can use the "View->Outline" menu in Microsoft Word to easily view and edit the testplan hierarchy.

Also, you can enable Word’s auto-numbering for the "Heading" styles so that the source document shows the numbers that will be used for the various sections in the imported plan.

- Add any number of paragraphs to each section or sub-section, using the "Normal" style. All paragraphs (up to the next "Heading" heading) are processed as part of that particular testplan section.

  - Enter a label (“<Name>:”) at the beginning of your “Normal” paragraphs. (See “Recognized Paragraph Labels in Word and DocBook” for a list of these labels.) Since a word-processor document does not have "columns", the data is identified by text labels embedded in the paragraphs. The label must always be the first non-blank text in the paragraph. It consists of the name of the data item followed by a colon. For example:

    DESCRIPTION: This is a description of section 1.3.5.

    Additionally, any paragraphs that begin with alphanumeric characters followed by a colon are considered a new data field in the testplan.

Guidelines for DocBook

DocBook is an open documentation format, based on XML. DocBook documents can be created and maintained using any number of specialized XML editors (like XmlMind's XXE or SyncRO Soft's Oxygen) and the format is simple enough to be edited with a standard text editor. DocBook, like Microsoft Word, supports a hierarchical document structure.

- Title your top-level testplan sections, using the “chapter” tag (style).

- Use the "section" style for any sub-sections under each section (that is: section "1.1" under section "1").
- Nest these verification plan sections to any desired level.

Tip: You need not enter section numbers when using this format, because the testplan import utility is configured to auto-number the sections according to the document hierarchy. The XXE editor uses stylesheets to render a formatted view of the document that includes automatically-generated section numbers.

- You can enter a "title" tag in each "chapter" or "section" tag, placing it as the first tag in the associated chapter or section. The text you use in this "title" tag becomes the testplan section name.
- Within each "chapter" or "section", following the "title" tag, you can enter any number of paragraphs marked with the "para" tag. The import utility processes these as part of the testplan section in which they are contained.
    - Enter a label at the beginning of each “para” paragraph. (See “Recognized Paragraph Labels in Word and DocBook” for a list of these labels.) Since a word-processor document does not have "columns", the data is identified by text labels embedded in the paragraphs. The label must always be the first non-blank text in the paragraph. It consists of the name of the data item followed by a colon. For example:

      DESCRIPTION: This is a description of section 1.3.5.

      Additionally, any paragraph that begin with alphanumeric characters followed by a colon are considered a new data field in the testplan.

Recognized Paragraph Labels in Word and DocBook

The following table lists the paragraph labels that are automatically recognized in Microsoft Word (and DocBook) by the importer:

Table 2-2. Recognized Paragraph Labels

Paragraph Label	Description
DESCRIPTION:	Free-form text description of the testplan section
GOAL:	Coverage goal
LINK:	Paths and/or match strings mapping the testplan section to specific coverage items in the design. For hierarchical testplans, you would specify “XML” here, in the parent testplan.
TYPE:	Type of link (either one type or one type per LINK entry)
WEIGHT:	Coverage weighting factor

These labels refer directly to the information contained in Table 2-1. For further details about each of these labels, see this table.

- When more than one "LINK" data item or "TYPE" data item is used, each text item should have a label. Multiple "LINK" data items are supported so that multiple coverage items can be linked with that testplan section.
- At least one "TYPE" data item is necessary before the "LINK" data item can correctly identify the type of coverage item that is defined in the following "LINK" data items. The "TYPE" data item is then sticky, and that coverage type is used for each "LINK" data item until another "TYPE" data item is defined. The exception to this rule is the case of Excel testplans, which require a “TYPE” entry for each “LINK” entry.
- The "GOAL" and WEIGHT" data items are also sticky and function much the same as “TYPE” and “LINK”. The “GOAL” defaults to “100”, and the “WEIGHT” defaults to “1” for each testplan item.

Related Topics

- Excel Add-In for Creating and Editing Testplans
- Importing an XML Verification Plan
- Exporting a Plan to XML
- About Hierarchical Testplans
- Importing a Hierarchical Plan
- XML Terms and Concept Review for Plan Import

Exporting a Plan to XML

The flow documented in this section was developed prior to the availability of the “Excel Add-In for Creating and Editing Testplans”. You are encouraged to try this new and improved flow over that contained in this section.

Before you can import a verification plan, if not using the Questa Excel AddIn program, the plan must be exported to XML format. Most documentation tools have some kind of XML export facility. The XML standard defines a general markup syntax through which the data in a given file may be annotated with meta-data. It does not, however, define the semantics of the meta-data (that is, the meaning of the markup tags). Each documentation tool has its own markup tags, each with its own unique semantics.

Examples of exporting plans from Excel, Word and DocBook are shown in this section.

Exporting From Microsoft Excel or Word

Prerequisites

- In order for the data to be properly interpreted by the import utility, you must have written the testplan following the guidelines listed in "Guidelines for Excel Spreadsheets" and "Guidelines for Word Documents," respectively.

Procedure

1. Open the spreadsheet or document.
2. Save as an .xml file:
    - Select **File** > **Save As**. This brings up the Save As dialog box.
    - In the **Save as type** pulldown menu, select **XML Spreadsheet (*.xml)**. This automatically changes the suffix of the spreadsheet or document to .xml.

Important: If using Excel 2007, you must select "XML Spreadsheet 2003" as the Save as type pulldown option. Note that this is different than the "XML Data" save option.

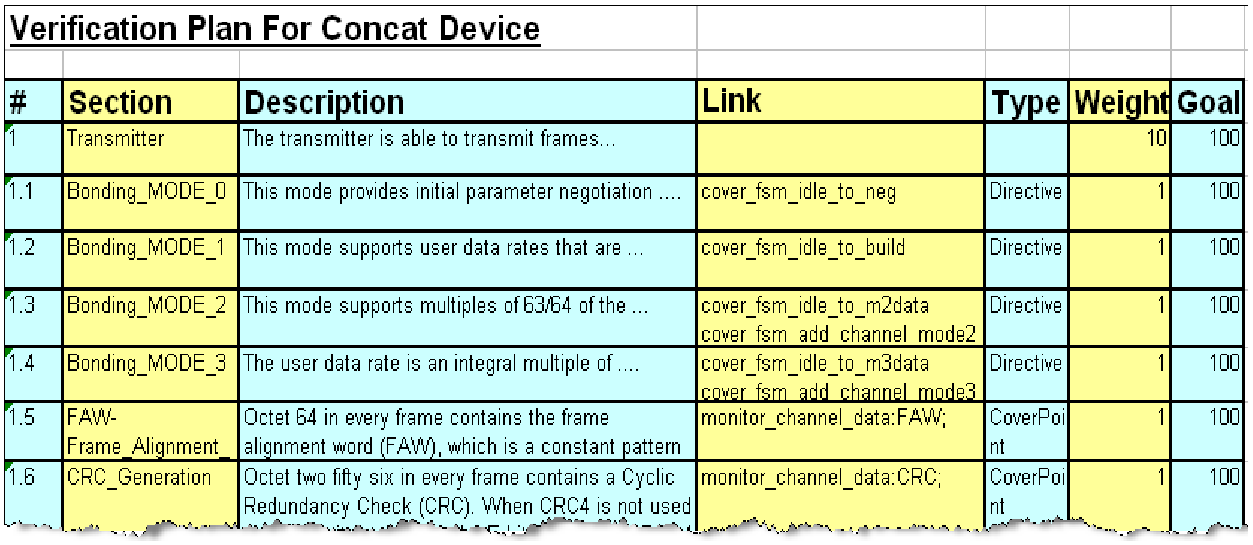
- Enter **Save**.

This saves the output as an .xml file.

Example 2-4. Verification Plan in Excel

An example verification plan spreadsheet written in Excel is shown in Figure 2-2.

Figure 2-2. Excel Spreadsheet Example Verification Plan



Verification Plan For Concat Device

#	Section	Description	Link	Type	Weight	Goal
1	Transmitter	The transmitter is able to transmit frames...			10	100
1.1	Bonding_MODE_0	This mode provides initial parameter negotiation	cover_fsm_idle_to_neg	Directive	1	100
1.2	Bonding_MODE_1	This mode supports user data rates that are ...	cover_fsm_idle_to_build	Directive	1	100
1.3	Bonding_MODE_2	This mode supports multiples of 63/64 of the ...	cover_fsm_idle_to_m2data cover_fsm_add_channel_mode2	Directive	1	100
1.4	Bonding_MODE_3	The user data rate is an integral multiple of	cover_fsm_idle_to_m3data cover_fsm_add_channel_mode3	Directive	1	100
1.5	FAW-Frame_Alignment	Octet 64 in every frame contains the frame alignment word (FAW), which is a constant pattern	monitor_channel_data:FAW;	CoverPoint	1	100
1.6	CRC_Generation	Octet two fifty six in every frame contains a Cyclic Redundancy Check (CRC). When CRC4 is not used	monitor_channel_data:CRC;	CoverPoint	1	100

Example 2-5. Excel's XML Output

Once you export the Excel spreadsheet verification plan into XML, the output appears as follows (some extraneous tags and tag attributes have been deleted to make the example easier to read):

```
...
<Worksheet ss:Name="Sheet1">
  <Table ...>
    ...
    <Row ...>
      <Cell ...><Data ...>Verification Plan For Concat
               Device</Data></Cell>
      <Cell .../>
    </Row>
    <Row .../>
    <Row ...>
      <Cell ...><Data ...>Item</Data></Cell>
      <Cell ...><Data ...>Test</Data></Cell>
      <Cell ...><Data ...>Description</Data></Cell>
      <Cell ...><Data ...>Link</Data></Cell>
      <Cell ...><Data ...>Weight</Data></Cell>
      <Cell ...><Data ...>Type</Data></Cell>
    </Row>
    <Row ...>
      <Cell ...><Data ...>1</Data></Cell>
      <Cell ...><Data ...>Transmitter</Data></Cell>
      <Cell ...><Data ...>The transmitter is able to
                           transmit frames...</Data></Cell>
      <Cell .../>
      <Cell ...><Data ...>1</Data></Cell>
      <Cell .../>
    </Row>
    <Row ...>
      <Cell ...><Data ...>1.1</Data></Cell>
      <Cell ...><Data ...>Bonding MODE 0</Data></Cell>
      <Cell ...><Data ...>This mode provides initial
                           parameter...</Data></Cell>
      <Cell ...><Data ...>cover_fsm_idle_to_neg</Data></Cell>
      <Cell ...><Data ...>1</Data></Cell>
      <Cell ...><Data ...>Directive</Data></Cell>
    </Row>
    <Row ...>
      <Cell ...><Data ...>1.2</Data></Cell>
      <Cell ...><Data ...>Bonding MODE 1</Data></Cell>
      <Cell ...><Data ...>This mode supports user data rates that are
                           multiples...</Data></Cell>
      <Cell ...><Data ...>cover_fsm_idle_to_build</Data></Cell>
      <Cell ...><Data ...>1</Data></Cell>
      <Cell ...><Data ...>Directive</Data></Cell>
    </Row>
    ...
  </Table>
</Worksheet>
```

The first "Row" in the spreadsheet contains the title for the document.

The second "Row" contains the column headings as found in the source document. The actual strings uses for these headings are irrelevant.

The actual data starts on the third "Row" of the spreadsheet, where the first column contains the literal string "1" (as specified in the “startstoring” parameter). From that point on, each "Row" in the spreadsheet is treated as a coverage section.

The "Cell" columns are assumed to occur in the sequence specified by the “datafields” parameter. The additional embedded "Data" tag is ignored. Once inside the tag specified by the “datatags” parameter, all child tags are ignored and only the marked-up data is used.

Related Topics

- Excel Add-In for Creating and Editing Testplans
- XML to UCDB Conversion Examples
- Guidelines for Writing Verification Plans
- Importing an XML Verification Plan
- Merging Verification Plan with Test Data

Exporting From DocBook

Since DocBook is a native XML format, it is not necessary to "export" the document to XML — simply import the document as-is.

Importing an XML Verification Plan

Once you have exported the verification plan into XML, you need to import the XML formatted file into UCDB for the purposes of merging the plan with your test data.

The xml2ucdb command along with the *xml2ucdb.ini* file is used by advanced users to control various aspects of the data extraction during verification plan import.

The extraction parameters used to import the verification plan into the UCDB are contained in a configuration file called *xml2ucdb.ini*. For any of the above supported formats, no change to this configuration file is needed. However, it can be helpful to view the file to get a better idea of how the conversion process works. See “xml2ucdb.ini Configuration File” for a snapshot of the file and more information on the format and syntax of the configuration file.

Supported Plan Formats

Questa SIM allows you to import your formal verification plan, documented in the following formats, into the UCDB:

- Microsoft Excel
- Microsoft Word

- DocBook
- FrameMaker
- GamePlan

Prerequisites

- For clean importation of the verification plan, write the original source verification plan in accordance with a set of guidelines specific to the accepted documentation formats (see "Supported Plan Formats" and "Guidelines for Writing Verification Plans").
- Export the spreadsheet/document to version 1.0 XML (inside your documentation tool). See "Exporting a Plan to XML".
- If you want to import one plan into another (termed "nesting"), you must edit the plans as shown in "Importing a Hierarchical Plan" before proceeding.

Importing Plan from the Command Line

From the command prompt, enter the xml2ucdb command and arguments to specify XML input file, the UCDB output file and other parameters for the conversion. For example:

```
xml2ucdb -format Excel input.xml output.ucdb
```

If you want to import an XML file residing in a directory other than the current directory (in which the xml2ucdb command is issued), you can specify the path using the -searchpath argument.

If your translation process requires customizing, edit the *xml2ucdb.ini* file. This file operates in conjunction with the xml2ucdb command to conform to any input format you might need. See "Customized Import with xml2ucdb.ini Config File" and "xml2ucdb.ini Configuration File" for usage and reference information on the configuration file.

Importing Plan from the GUI

1. Select **View** > **Verification Management** > **Browser**.
2. Right-click anywhere inside the Verification Browser window and select **Test Plan Import**.

   This displays the XML Verification Plan Import Dialog Box. The options within the dialog box correspond to the arguments available with the xml2ucdb utility.
3. Specify the **XML Input File**.
4. Select the file format from the **Format** drop down list.
5. Specify the name and location for the generated **UCDB Output File**.

6. If you want to automatically create a script file that contains the mapping of the data from the XML to the UCDB, select the **Generate tagging script** checkbox and specify the name and location for the generated file.

Tip: Because the tags are stored in the UCDB as part of the import process, this file is not required for normal operation. However, it can be useful in determining how the testplan maps to the design, although it can result in additional overhead.

7. Click **OK**.

   The XML is imported and converted to UCDB, recording the testplan associations in the UCDB.

   At this point, you can merge the test (verification) plan with your test data, thereby associating the items from the testplan with coverage points in the design. See "Links Between the Plan Section and the Coverage Item" and "Merging Verification Plan with Test Data" for more information.

Results

When you have successfully written out the testplan to UCDB format, a message is displayed similar to the following:

```
# Format 'Excel' read from parameter file
# Database 'testplan.ucdb' written.
```

If errors occur during importation, due to deviations in the verification plan from the "Guidelines for Writing Verification Plans", give careful consideration to modifying the extraction parameters in the *xml2ucdb.ini* file to suit the specifics of your verification plan. See "xml2ucdb.ini Configuration File" for reference information on the configuration file.

Related Topics

- XML to UCDB Conversion Examples
- xml2ucdb Command
- xml2ucdb.ini Configuration File

Importing a Hierarchical Plan

A verification plan can contain hierarchy, whereby other testplans are "nested" within a top level verification plan. The numbering of sections within a hierarchical plan is handled as if the contents of the "child" file were pasted into the "parent" plan section. The instructions for handling section numbering depend on whether the plans being imported are autonumbered (as in Word) or non-autonumbered (spreadsheet):

- If both "parent" and "child" use auto-numbering, imported sections are numbered as children of the importing testplan section.

- If importing a mixture of autonumbered and non-autonumbered plans, consult "Mixing Autonumbered and Non-autonumbered Plans" before proceeding.
- If importing non-autonumbered testplans, or importing a plan section from one verification plan under a specified subsection of the main verification plan, consult "Numbering of Non-autonumbered Nested Plans" before proceeding.

The instructions in the remainder of this section apply to the importation of autonumbered (Word) plans.

Prerequisites

- If unfamiliar with hierarchical ("nested") plans and their specific requirements, review "About Hierarchical Testplans".
- Familiarize yourself with limitations to the importation of nested plans, see "Limitations".

Procedure

1. Identify the plan which serves as the parent.
2. Enter the following information into the parent XML plan:
   a. In the **Type** field (or column) of the parent XML plan, enter the string "XML".
   b. In the **Link** field (or column) of the parent XML plan, enter the child XML file. For example:

      [<path>]/<child1>.xml

Note

Alternatively, you could specify the name of the file to import in the **Link** field, and the path to that file in the **Path** field. If no path is specified in either field, the current directory is assumed.

3. Import the nested testplan, just as described in "Importing an XML Verification Plan". There is no difference in the actual importation process: just in the setup of the files themselves.

Related Topics

- About Hierarchical Testplans
- xml2ucdb.ini Configuration File
- xml2ucdb Command

Adding Columns to an Imported Plan

Questa SIM requires that each plan being imported contain the pre-defined attributes defined in Table 2-1. Each attribute corresponds to a column with the same name in your verification plan.

You can add additional attributes to an imported UCDB file, to match the columns as they exist in your individual verification plan. This information can be used in later analysis to filter coverage results, once the plan has been imported and merged with the test runs. See “Filtering Results by User Attributes” for instructions.

To customize the list of columns that appear in the plan, you must modify the “datafields” in the plan manually, by either:

- using xml2ucdb -datafields <all fields> at command line, where <all fields> is an ordered list of all data fields in the plan, including the fields you wish to add. These are separated by whitespace, and are listed in the order they appear in your plan. This adds the data field only to the specific UCDB being imported.
- copying *xml2ucdb.ini* (located in *install_directory/vm_src*) into your own directory and hand editing the file. This adds the data field(s) to all imported verification plans that will utilize this configuration file.

Customized Import with xml2ucdb.ini Config File

The *xml2ucdb.ini* configuration file, as shipped with the tool, is sufficient for the vast majority of all imports. Generally, you should not need to modify this file to import a verification plan if the verification plan (top level testplan) was created:

- using one of the tools listed in “Supported Plan Formats”, and
- following the guidelines listed for each format in the “Guidelines for Writing Verification Plans”.

However, verification plans written outside the scope of these guidelines may require you to customize the file's extraction parameters.

The *xml2ucdb.ini* file contains the extraction parameters for the conversion of an XML file to UCDB. It is located in the *<install_dir>/vm_src/* directory.

You can override the default *xml2ucdb.ini* file by placing a file with the same name in the current working directory. The file is in standard INI format (http://en.wikipedia.org/wiki/INI_file). See "xml2ucdb.ini Configuration File" for more information on this file.

Excluding Items from a Testplan

Several methods exist for excluding information that exists in a testplan from the UCDB, depending on what you want to exclude:

- Excluding Linked Coverage with LinkWeight — specify 0 in LinkWeight column
- Excluding a Testplan Section Using the Weight Column — specify 0 in Weight column
- Excluding Portions of a Plan — using "-excludetags" parameter in configuration file

When Should Items be Excluded from a Testplan?

During the process of developing a testplan, you may not want to register a low coverage score, such as when you have unimplemented testplan sections.

Another reason you might want to exclude testplan items is that often coverage models must be written for a superset of all possible functionality. One example of this is Questa Verification IP, which provides all possible functionality associated with a specific protocol. Yet a user might only want to use a subset of that protocol. In such cases, the unused portions of the protocol coverage model must be excluded, or else the coverage score will be artificially low. Zero weights can be used in the testplan to effectively exclude the undesired sections.

Excluding the coverage from testplan coverage calculations can work for either of the two above scenarios.

Excluding Linked Coverage with LinkWeight

Several methods exist for excluding individual coverage items, or excluding a subsection of items. By way of illustrating the various methods discussed in this section, consider the verification plan shown in Table 2-3:

Table 2-3. Plan Weight and LinkWeight Usage for Excluding Coverage

Testplans	Link	Weight	LinkWeight
Section1	cov1	1	1
Section2	cov2	1	1
	cov3		0
	cov4		1
Section3	cov5	0	1
	cov6		0

Table 2-3. Plan Weight and LinkWeight Usage for Excluding Coverage

Testplans	Link	Weight	LinkWeight
Section4	cov5	1	0
	cov6		

To exclude linked coverage, whether an item or a subsection of items, you must:

1. Add a column called LinkWeight to your testplan.
2. Add the LinkWeight column specification within the -datafields parameter of the *xml2ucdb.ini* file, ensuring it is placed in the order that matches its placement within your testplan. See "Parameter for Mapping by Column Sequence" for details on parameter placement.

   To exclude a single item in a testplan, or all items in a testplan using LinkWeights, you must also:

   - Set the value in the LinkWeight column to 0 for each individual coverage items.

   Or, to exclude one or more of multiple coverage items in a verification plan row (see *Section2* in Table 2-3), you can either:

   - enter a single 0 value in the LinkWeight column — which applies to all items listed in that row (see row *Section4*). As such, any coverage items in that section is excluded from the testplan coverage score.
   - enter the same number of 0 values (or as there are coverage you want to exclude in that row (in this case three) within the LinkWeight column. This way, you can exclude linked coverage items from the testplan coverage score, on an individual basis.

If no link-specific weight is assigned through use of the LinkWeight column, the regular weighting rules apply, as discussed in "Coverage Calculation in Testplans". However, when LinkWeight is used, the weight for each link is also rolled into the testplan coverage calculation.

Excluding a Testplan Section Using the Weight Column

To exclude a testplan, set the value of the Weight column for a testplan section to 0 within the overall testplan. See "Coverage Calculation in Testplans" for details on weighting and calculations.

Excluding Portions of a Plan

To exclude whole portions of the plan, before the testplan is imported, using the "excludetags" parameter:

1. Define the input you want to exclude using XML tags within the verification plan.

2. Specify the XML tag associated with the data you wish to exclude from the UCDB by:
    - adding the "excludetags <tags>" parameter to a customized *xml2ucdb.ini* file (see "Parameters Controlling the Inclusion of Data")
    - adding the "-excludetags <tags>" to the xml2ucdb command

All data defined within the specified tags will be excluded from the UCDB when it is imported.

By doing so, you define the XML tags that will appear around parts of the input that you want to ignore. This could be set to the XML tag name that the document uses for the images, it can also take a list of tags for multiple tag names.

About Hierarchical Testplans

A hierarchical (or nested) testplan is a testplan which is imported within an overall testplan UCDB file. Only one UCDB (containing the nested testplan) is generated per importation. The parent plan may be referred to as a "hierarchical" plan.

Testplan nesting is not limited to one level. A nested testplan can, itself, contain other nested plans (to the limit of one's sanity and the machine's memory). For any given nesting event, "child" refers to the file being nested, while "parent" refers to the file into which the child's sections are imported.

Mixing Autonumbered and Non-autonumbered Plans

Section numbering is handled as if the contents of the "child" file were pasted into the "parent" testplan section. The following rules and limitations apply when mixing autonumbered and non-autonumbered plans.

- If both "parent" and "child" use auto-numbering, imported sections are numbered as children of the importing testplan section, as expected.
- If the top-level file uses explicit numbering, any "child" files that use auto-numbering are NOT numbered (this may result in an incorrect UCDB file).
- If the top-level file uses auto-numbering, the entire plan (including explicitly numbered sections from "child" files) is renumbered.

Numbering of Non-autonumbered Nested Plans

When a nested plan is imported, that section in the parent plan (in which the "XML" Link type is specified) becomes the "root" level for that plan section. All top-level sections of the nested verification plan become children of this section.

To set the number for the root of the nested (child) verification plan being imported, use the -root argument in the **Link** field. An example would be:

-root <section_number> <child>.xml

where <section_number> is the number that becomes the root number within the parent.

Note

The “-root” argument is not necessary when importing an auto-numbered (word-processor based) child verification plan.

The "-root" argument should only be used if the section numbers in the child testplan differ from those inside the parent file. For example, if you nest a child with sections “4.1, 4.2, 4.3, and so forth," into section 4 of the parent, the "root" option is not needed; the root number “4” matches within the two files. If, on the other hand, the child testplan has sections "1, 2, 3, and so forth," and you import them into section 4 of the parent, specifying a "-root 4" argument imports the child section "1" as "4.1", thus ensuring correct numbering.

Example of a Nested Plan

Following is an example of a nested verification plan. Assume you have the following two testplans: the “parent” plan shown in Table 2-4 and the “child” plan in Table 2-5:

Table 2-4. parent.xml

Section	Title	Type	Link
1	ParentOne		
1.1	ParentOneDotOne		
2	ParentTwo	XML	-root 2 child.xml
3	ParentThree		
3.1	ParentThreeDotOne		

When these plans are imported, the parent testplan sections are stored under a root testplan section numbered "0" with the title "testplan". The child testplan's sections are imported into the UCDB as children of the parent section "2" because the parent section "2" contains the XML-type Link. This Link entry is what causes the nested plan to be imported. The "-root 2" option causes the child testplan section numbers to be prepended with the string "2.".

Table 2-5. child.xml

Section	Title
1	ChildOne
1.1	ChildOneDotOne
2	ChildTwo
2.1	ChildTwoDotOne

The resulting UCDB testplan tree is shown in Table 2-6:

Table 2-6. Nested Testplan

Section	Title
0	testplan
1	ParentOne
1.1	ParentOneDotOne
2	ParentTwo
2.1	ChildOne
2.1.1	ChildOneDotOne
2.2	ChildTwo
2.2.1	ChildTwoDotOne
3	ParentThree
3.1	ParentThreeDotOne

Inheritance in Hierarchical Plans

You can specify that the nested testplan inherit the parent's storing state (resulting from start/stop tags or startstoring tag) by using the -inherit argument. See xml2ucdb for more information. The reasons to use -inherit, and the concepts behind it, are as follows.

Sometimes, especially when the parent and child are both Excel documents, a parent may have a "startstoring" parameter set so that the extraction of testplan information doesn't start until a row with a specific section number (usually "1"). . If the child testplan has been renumbered to match the parent section into which it's imported (that is: sections 4.1, 4.2, 4.3, and so forth), the "startstoring" section number is never seen in the child, since there is no section "1". In this case, nothing will be imported.

One way around this would be to re-define "startstoring" for the child. However, if the parent "startstoring" section had not yet been seen, you would not want the child imported anyway. Moreover, the same thing happens with the "stoptags" and "starttags". If the parent importing a child appears between a stop and start tag, you might also want the child to honor that blocked section.

A smoother solution in these cases would be to inherit the state of the storing flags from the parent to the child importation process. Using the "-import" option on the command in the **Link** line of the parent section causes the child importation to begin with the parent's storing state values. The -import option has no effect when used on the top-level import command line.

Limitations

A few limitations exist for nested testplan importation, as follows:

- The XML-type link must be the only link in the testplan section where the "child" testplan is to be imported.
- Parameter inheritance (see “Inheritance in Hierarchical Plans”) from parent to child may cause the child import to pick up undesired extraction parameter values from the parent. If this happens, use the relevant extraction parameter to override the unwanted parent setting.
- The -help, -debug, -verbose, -viewtags, and -viewall arguments to the xml2ucdb import command are global. If specified in the link field of an XML-type testplan section, the arguments are activated from that point to the end of the import process.
- Arguments that specify the parent testplan file or the UCDB output file (that is, -dofilename, -ucdbfilename, and so forth) have no effect if specified in the link field of an XML-type testplan section.

XML to UCDB Conversion Examples

This section describes the XML format of some common documentation tools. It also details the specific settings required to capture the data from XML files generated by that specific tool.

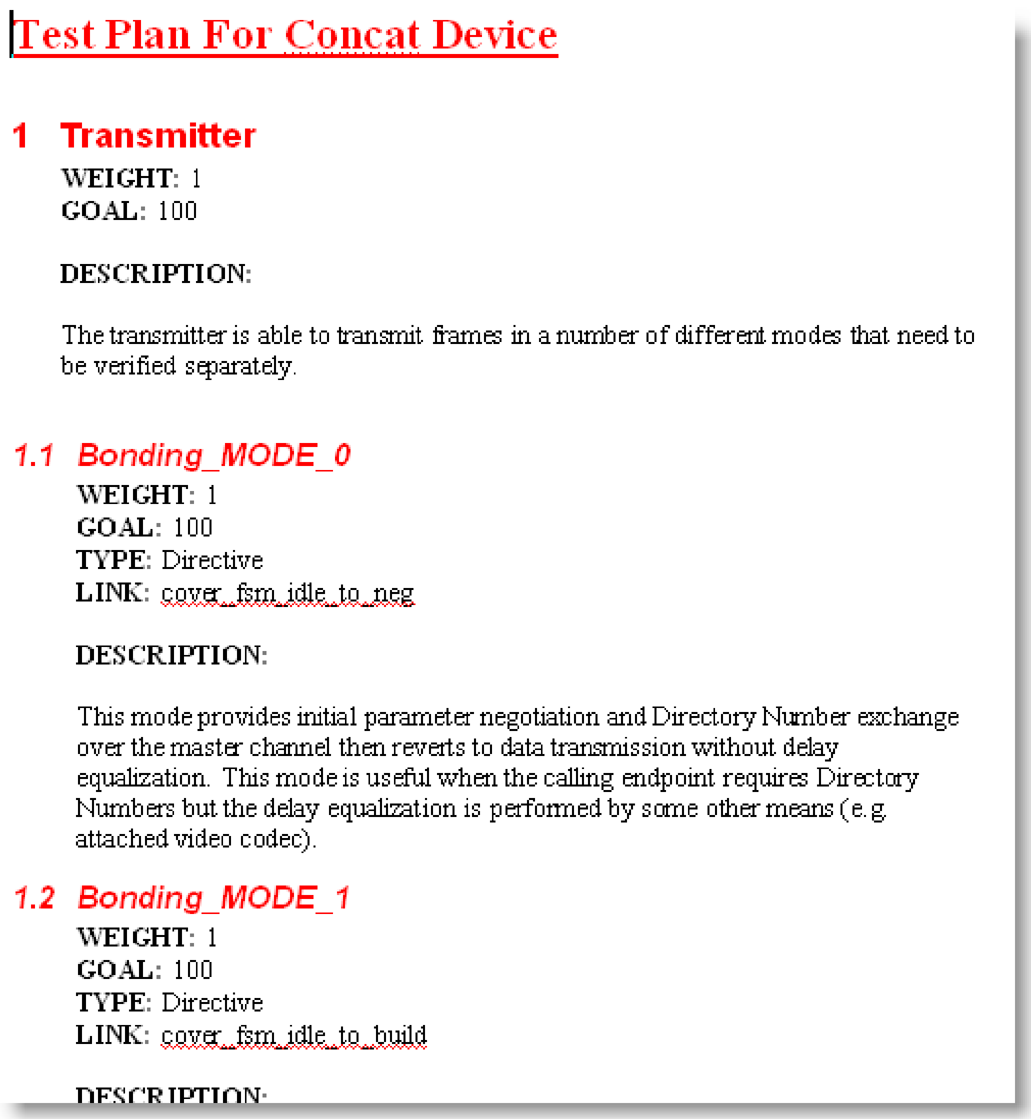
Microsoft Word Example

This section contains an example excerpt from a verification plan written in Microsoft Word and exported in XML format.

Source Document for Word

A sample from a Word-based version of a verification plan document:

Example 2-6. Partial Source Document for Word


Test Plan For Concat Device

1 Transmitter

WEIGHT: 1
GOAL: 100

DESCRIPTION:

The transmitter is able to transmit frames in a number of different modes that need to be verified separately.

1.1 Bonding_MODE_0

WEIGHT: 1
GOAL: 100
TYPE: Directive
LINK: cover_fsm_idle_to_neg

DESCRIPTION:

This mode provides initial parameter negotiation and Directory Number exchange over the master channel then reverts to data transmission without delay equalization. This mode is useful when the calling endpoint requires Directory Numbers but the delay equalization is performed by some other means (e.g. attached video codec).

1.2 Bonding_MODE_1

WEIGHT: 1
GOAL: 100
TYPE: Directive
LINK: cover_fsm_idle_to_build

DESCRIPTION:

The text items marked with a number, as in "1," “1.2,” “1.2.3," use the "Heading" style appropriate to the level of hierarchy they occupy in the verification plan. All other text items are marked as "Normal" style, although other styling options may be applied according to the user's aesthetic sensibilities. Also, any additional styling applied to the text items, headings, or data item labels is ignored.

DocBook Example XML File

The following is an example of a DocBook XML file:

```
    <?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
"http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd">
```

```
<book>
  <title>testplan</title>
  <chapter>
    <title>Transmitter</title>
    <para><emphasis role="bold">WEIGHT:<emphasis>
                              1</emphasis></emphasis></para>
    <para><emphasis role="bold">GOAL:</emphasis> 100</para>
    <para><emphasis role="bold">DESCRIPTION:</emphasis></para>
    <para>The transmitter is able to transmit frames...</para>
    <section>
      <title>Bonding_MODE_0</title>
      <para><emphasis role="bold">WEIGHT:</emphasis> 1</para>
      <para><emphasis role="bold">GOAL:</emphasis> 100</para>
      <para><emphasis role="bold">TYPE:</emphasis> Directive</para>
      <para><emphasis role="bold">LINK:</emphasis>
                              cover_fsm_idle_to_neg</para>
      <para><emphasis role="bold">DESCRIPTION:</emphasis></para>
      <para>This mode provides initial parameter negotiation...</link>
    <section>
      <title>Bonding_MODE_1</title>
      ...
    ...
  ...
</book>
```

The "emphasis" tags, like all other visual styling markup, are ignored by xml2ucdb. Only the text content is relevant.

Combining Verification Plan Data and Test Data

Prerequisites

Before you can combine your verification plan data with the data from tests you perform you must have already:

1. Written a verification plan — For the smoothest importation, the verification plan should be written in accordance with the guidelines outlined in “Guidelines for Writing Verification Plans”.

2. Exported plan to .xml format within tool used to write plan — See “Exporting a Plan to XML”.

3. Imported .xml formatted plan into a .ucdb — See "Importing an XML Verification Plan".

Merging Verification Plan with Test Data

The real power behind importing your verification plan to UCDB lies in the linking of the sections in the verification plan to specific coverage items in the design. This link is forged when you merge your verification plan (top level testplan) with the coverage test data in UCDBs, taken from your test runs. During this process, the specific verification plan items are linked to coverage items automatically, assuming that the testplan UCDB was generated by either the Test Plan Import facility or the xml2ucdb command (see "Importing an XML Verification Plan"). Because of the way the testplan UCDB is annotated by the import utility, the "coverage tag" command runs implicitly during the merge and links testplan sections with the covergroups. Objects which share the same tag name are thereby linked together.

Requirements

In order to merge your verification plan with the UCDBs from your test runs, you must do the following:

1. Import your verification plan into a UCDB format. See "Importing an XML Verification Plan" for detailed instructions.
2. Merge the verification plan UCDB with the test run UCDB(s) using the same methods you use to merge test data:
   - with the GUI (see "Merging within Verification Browser Window")
   - with vcover merge (see "Merging with the vcover merge Command")

A merged UCDB containing a verification plan is viewable in the Verification Tracker window when invoked in Coverage View mode.

Related Topics

- Merging Coverage Data
- Warnings During Merge
- Merge Usage Scenarios
- About the Merge Algorithm

What are Unimplemented Links and Why Use Them?

During the process of developing a testplan, you may determine that some of your testplan sections have linked coverage objects which are not yet implemented (still under construction, or otherwise missing). However, you may want to see an accurate and complete picture of your coverage as it exists at that time. By default, whenever a link is missing, the coverage objects associated with that link are not counted in the calculation of coverage numbers. This has the effect of artificially increasing your coverage percentages, which can give you a false sense of security.

The solution is to define items with missing links as an “Unimplemented”, using the methods detailed in “Counting Coverage Contribution from Unimplemented Links”. This will automatically generate zero coverage virtual objects for these items and link them to the appropriate section(s) of the testplan. Thus, the linking of these 0% unimplemented coverage objects causes the overall coverage number to be reduced, which more accurately reflects the fact that more work is required toward your end goal.

Counting Coverage Contribution from Unimplemented Links

To count incomplete or unimplemented coverage item(s) within a testplan, you can choose one of two methods. Each method has certain advantages.

- xml2ucdb -createcovitem —

  When this xml2ucdb switch is used, one coverage bin will be created for each coverage item not found in the coverage database when the testplan is merged with a coverage UCDB. This method has the advantage that it is easy to use: you don’t need to add an additional column to your testplan.

- Unimplemented column —

  When column is added to testplan, it creates the specified number of empty coverage bins. This method allows control over the number of empty bins created for individual testplan and testplan subsection links.

Procedure for Adding Unimplemented Column

1. Add a column called “Unimplemented” to your testplan.
2. Add the Unimplemented column specification to the -datafields parameter within the *xml2ucdb.ini* file, ensuring it is placed in the order that matches its placement within your testplan. See “Parameter for Mapping by Column Sequence” for details on parameter placement.
3. Set the value in the Unimplemented column. Possible values are “No” = 0, “Yes” = 1, or <int> = multiple:
   - Yes or “1” - creates a single empty bin.
   - Or, if your item when implemented will create multiple bins, and you want to register the effect those bins would have, specify an <integer> number of bins to create.

   This instructs the tool to create the designated number of empty coverage bins, which will contribute to the coverage total.

Linkable Design Constructs

Table 2-7 lists the various constructs within a design, to which a testplan section can be linked. These construct types are placed in the "Type" field within the verification plan.

For a list of fields/columns that Questa SIM expects to exist in the plan, see Table 2-1.

Important: Do not abbreviate the names of the coverage constructs listed in Table 2-7.

Table 2-7. Recognized (Linkable) Coverage Design Constructs

Coverage Construct — in "Type" field (case insensitive)	**Description**	**Syntax (follow the links)**
Assertion	Assertion statement	Assertions, Directives, and Generic Coverage Items
Bin	Coverage item bin	Bin Links
Branch	Branch coverage scope	Code Coverage Links
Condition	Condition coverage scope	Code Coverage Links
CoverGroup	SystemVerilog covergroup statement	Covergroups, Coverpoint and Crosses
CoverPoint	SystemVerilog coverpoint statement	Covergroups, Coverpoint and Crosses
CoverItem	Generic name for any coverage or design object in a UCDB. This can be used to specify any objects not fitting into another category of construct.	Assertions, Directives, and Generic Coverage Items
Cross	SystemVerilog cross-coverage statement	Covergroups, Coverpoint and Crosses
Directive	PSL cover directives and SystemVerilog "cover" statements/properties	Assertions, Directives, and Generic Coverage Items
DU	All coverage on a given design unit	Instances and Design Units Links
Expression	Expression coverage scope	Code Coverage Links
FSM	State Machine coverage scope	Code Coverage Links
Instance	All coverage on a given instance	Instances and Design Units Links

Table 2-7. Recognized (Linkable) Coverage Design Constructs

Coverage Construct — in "Type" field (case insensitive)	**Description**	**Syntax (follow the links)**
Tag	Forms a link using any coverage tag command arguments which are specified in the Link column.	see "coverage tag" command for syntax
Test	Link to test attribute record. This is the test name.	Directed Test Links through Test Records
Toggle	Toggle coverage scope	Code Coverage Links
XML	Triggers hierarchical (nested) testplan import. See "About Hierarchical Testplans"	XML Links

Related Topics

- Links Between the Plan Section and the Coverage Item
- Syntax for Links to the Coverage Model
- Merging Verification Plan with Test Data
- Finding Unlinked Items
- coverage tag command
- Parameters Mapping Testplan Data Items

Links Between the Plan Section and the Coverage Item

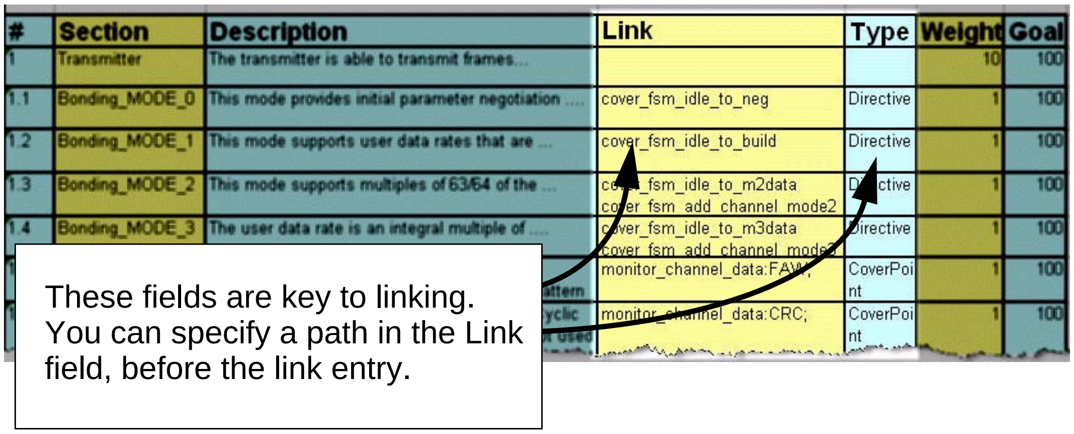
Linking allows you to verify via the testplan that coverage items are being covered. The links are performed automatically when you merge an imported testplan with actual tests you have run (see "Merging Verification Plan with Test Data"). Key to this linking, however, is the correct association of the design constructs — various linkable constructs in the design — to the verification plan items (see Backlink Coverage Items to Plan with SV Attributes

A link between a coverage item and a testplan section is accomplished, essentially before the plan is imported, using one of the following methods:

- For spreadsheets — by entering the coverage item name into the "Link" column/field, and one of the coverage construct keywords listed in the table below into the "Type" column/field (see Figure 2-3). Table 2-1 lists a the default name for columns/fields that can be linked.
- For word-processing documents — by placing the coverage item name using the "Link:" label, and placing one of the listed design constructs into using the "Type:" label. Table 2-2 lists the names of all labels recognized by the linking mechanism.

Tip: For advanced usage, you can also create or modify these links manually with the coverage tag command.

Figure 2-3. Verification Plan Linking



#	Section	Description	Link	Type	Weight	Goal
1	Transmitter	The transmitter is able to transmit frames...			10	100
1.1	Bonding_MODE_0	This mode provides initial parameter negotiation	cover_fsm_idle_to_neg	Directive	1	100
1.2	Bonding_MODE_1	This mode supports user data rates that are	cover_fsm_idle_to_build	Directive	1	100
1.3	Bonding_MODE_2	This mode supports multiples of 63/64 of the	cover_fsm_idle_to_m2data cover_fsm_add_channel_mode2	Directive	1	100
1.4	Bonding_MODE_3	The user data rate is an integral multiple of	cover_fsm_idle_to_m3data cover_fsm_add_channel_mode3	Directive	1	100
		attern	monitor_channel_data:FAW;	CoverPoint	1	100
		yclic used	monitor_channel_data:CRC;	CoverPoint	1	100

These fields are key to linking. You can specify a path in the Link field, before the link entry.

Datafield Rules and Requirements for Links

The rules for links between testplan sections and the design vary slightly depending on the design construct involved. Specifically, correct linking relies on your testplan containing the specific datafields relating to the columns in the verification plan. These datafields — defined in the *xml2ucdb.ini* configuration file — are used during import to correctly map the XML version of your verification plan to a UCDB format. See "Parameter for Mapping by Column Sequence" for further details.

The following datafields (at minimum) must exist in the *xml2ucdb.ini* for proper linking:

- Link — This is the name of (or path to) the actual design construct to which the testplan section is to be linked. If path separator (/) is used, everything placed in the entry prior to the final "/" is interpreted as the path.
- Type — This specifies what kind of design construct is being linked (see Table 2-7). The rules for linking testplan sections to design constructs vary depending on the type of design construct involved. See "Syntax for Links to the Coverage Model".
- Path — This field allows you to specify an optional string to limit any search for design constructs to a specific sub-hierarchy in the design. This is useful if, for example, there are multiple design constructs of the same name in various parts of the design and the testplan section is only supposed to be linked to some but not all of those similarly-named constructs.

Specifying Levels of Hierarchy in Plan

To specify the level of hierarchy for the design item you are linking, you can either:

- Enter the level of hierarchy in the “Path” field in the plan.
- Specify the path to the coverage item in front of the item name in the “Link” field. See Figure 2-3.

Related Topics

- Backlink Coverage Items to Plan with SV Attributes
- Finding Information for Missing Verification Plan Links
- Finding Unlinked Items
- Backlink Coverage Items to Plan with SV Attributes
- coverage tag command
- Parameters Mapping Testplan Data Items

Weighting of Unimplemented Testplan Sections

During the process of developing a testplan, you may have sections that are linked to coverage that has yet to be implemented. By default, any testplan item that is not fully implemented in the coverage model will be un-linked and will thus have no effect; only implemented items have an effect on the coverage calculation of the testplan.

However, if you do want your unimplemented coverage items to have a negative effect on the coverage numbers, you can make use of a facility that automatically generates coverage in the UCDB and links to it. Coverage numbers are then generated based on the link name. This ensures that you do not get a false sense of security when you see a score of 100%, yet you have unimplemented coverage linked to testplan sections. See “Counting Coverage Contribution from Unimplemented Links” for details and instructions.

Weighting to Exclude Testplan Sections from Coverage

Sometimes, you may not want to get a low testplan coverage score when you have unimplemented testplan sections. When this is the case, you can apply a weight of zero to such sections within the testplan. You can also use a weight of zero in any linked-to coverage items. Such zero-weighted coverage items will roll up into the testplan coverage score, causing their value to be excluded.

There is another reason you may want to exclude testplan items: Often coverage models have to be written for a superset of all possible functionality. An example of this is Questa Verification IP, which provides all possible functionality associated with a given protocol. Yet a user may only wish to use a subset of that protocol. In such cases, the unused portions of the protocol coverage model must be excluded, else the coverage score will be artificially low. Zero weights can be used in the testplan to effectively exclude undesired testplan sections. See “Excluding a Testplan Section Using the Weight Column” for instructions.

Backlink Coverage Items to Plan with SV Attributes

You may wish to link your coverage items to verification plan data within the code itself, as opposed to linking your coverage through entries in the plan. This is referred to as "backlinking" your data, and is accomplished through the use SystemVerilog attributes. These attributes are part of the language itself, and can be used for backlinking the following items:

- Covergroup, coverpoint, and cross declarations in SystemVerilog (not including covergroup variable declarations).
- Assert and assume statements in SVA (SystemVerilog)
- Cover statements in SVA (SystemVerilog)
- Immediate assert statements in Verilog
- Module, package, interface, and program declarations in SystemVerilog
- Module, program, and interface instantiations in SystemVerilog

Backlinking of branch, condition, expression, toggle and FSM scopes are not supported.

Attribute Syntax for Backlinking

The basic attribute syntax involves a specially purposed identifier, "tplanref", whose value is a comma-separated list of tag strings.

For example, in the following (Verilog) code:

```
(* tplanref = "testplan.1.2, subplan.2.3, testplan.2" *) covergroup ct;
```

the covergroup "ct" is backlinked to three different plan sections: "testplan.1.2", "subplan.2.3", and section "testplan.2". Certain characters are not allowed in the tag values: commas (","), open or closed parentheses ("(" and ")"), and whitespace (" "). The comma is reserved as a list delimiter and parentheses are reserved for the covergroup instance syntax (see Covergroup Instance Syntax and Example). Whitespace is allowed after the comma delimiter.

Attribute values must be elaboration-time constants in SystemVerilog. This does allow some forms of expressions including "?:" and concatenation, so long as terms of the expression are themselves elaboration-time constants. For example:

```
parameter inst = 0;
parameter item = 0;
parameter testplan = "testplan";
(* tplanref = inst ? "mytestplan.1" : { testplan, ".1.", item } *)
```

Module and Module Instance Syntax and Example

Specification of a tplanref in a module declaration links the aggregated design unit coverage for all forms of coverage found within the design unit. This is the equivalent of linking the design unit node in the UCDB with the testplan section.

Specification of a tplanref in a module instantiation links the recursively aggregated coverage for all forms of coverage found within that instance and all its sub-instances. This is the equivalent of linking the instance node in the UCDB with the testplan section.

Example 2-7. Backlinking Module Declaration

```
(* tplanref = "testplan.1.2 *)
module concat53(input next_state, frame_v, output alarm);
   ...
endmodule
```

Example 2-8. Backlinking Module Instances to a Plan Section (Verilog)

```
(* tplanref = "testplan.10.1" *) concat CHIPBOND (
                 .RESET(RESET),
                 .CLOCK(CLOCK),
                 .ADDRESS(ADDRESS[3:0]),
                 .PDATA(PDATA),
                 .RDB(RDB),
                 .CSB(CSB),
                 .WRB(WRB));
```

Covergroup Instance Syntax and Example

There is an additional syntax for links from a covergroup instance or a coverpoint or cross of a covergroup instance. Without this syntax, the link from a covergroup is considered to be for the type as a whole, and the covergroup type coverage will appear with the verification plan.

A name in parentheses (*(<instancename>)*) designates that all following tag names be associated with that instance only. Consequently, if you want to combine type-based covergroup linking and instance-based covergroup linking, the covergroup type-based linking must occur first in the tplanref value.

Example 2-9. Backlinking Covergroups and Coverpoints

```
covergroup monitor_channel_data (ref faw_t FawState, ref crc_t CRCState,
      ref persist_state_t GroupState, ref persist_state_t ChannelState,
      ref state_t FrameState, ref state_t InfoState,
      input string inst_name ) @(posedge VARIABLE_WRB);
  (* tplanref =  "testplan.10.2" *) FAW : coverpoint
      FawStateSample(FawState, inst_name) iff (RESET)
  {
    bins OUT_OF_SYNC  = { 0 }; bins SEEN_ONCE    = { 1 };
    bins SEEN_TWICE   = { 2 }; bins IN_SYNC      = { 3 };
    bins MISSED_ONCE  = { 4 }; bins MISSED_TWICE = { 5 };
    bins others = default;
  }
  CRC : coverpoint CRCStateSample(CRCState, inst_name) iff (RESET)
  {
    bins ENABLED       = { 0 }; bins SEEN_ONCE     = { 1 };
    bins SEEN_TWICE    = { 2 }; bins DISABLED      = { 3 };
    bins others = default;
  }
  (* tplanref =  "(chana)testplan.10.5,(chanb)testplan.10.6"  *) CHANNEL :
      coverpoint ChannelStateSample(ChannelState,inst_name) iff (RESET)
  {
    bins SEARCH      = { 0 }; bins MATCH       = { 1 };
    bins FOUND_TWICE = { 2 }; bins FROZEN      = { 3 };
    bins others = default;
  }
```

The above example demonstrates linking to covergroup instances (when in covergroup scopes), or coverpoints or crosses of instances (if in coverpoint or cross scope). The name within parentheses must be the same as the *option.name* for the covergroup, coverpoint, or cross instance.

Assertion Syntax Example

```
(* tplanref = "testplan.10.3" *) assert property (loc_full(FSCS,LOCN_FULL));
```

Cover Directive Syntax Example

```
(* tplanref = "testplan.10.3" *) cover property (loc_full(FSCS,LOCN_FULL));
```

Notes and Limitations on Backlinking

There is no way to link specific FSMs, toggles, or other individual code coverage UCDB nodes with SystemVerilog attributes. This is a limitation compared to forward linking.

Related Topics

- Merging Verification Plan with Test Data
- Finding Unlinked Items
- Links Between the Plan Section and the Coverage Item
- Finding Information for Missing Verification Plan Links

Finding Unlinked Items

To verify that the coverage of your verification plan and design items are covered as you expect, it is helpful to know what items in your verification plan are not associated with any item in the design database, and vice versa. Analysis of your test traceability relies on links between some aspect of coverage and sections of a testplan. These links are implemented with the coverage tag command: objects which share a tag are linked.

The following items may be found to be unlinked:

- unlinked testplan item — one that does not share a tag with any non-testplan item in the database; this means it cannot possibly be associated with any kind of coverage
- unlinked functional coverage item — one that does not share a tag with a testplan section

Procedure

To find unlinked testplan or coverage items:

- In the GUI:
    a. Select a single UCDB file from the Tracker.
    b. From the main menu, select **Verification Tracker > Find Unlinked** or in the Verification Tracker window: Right mouse button > **Find Unlinked > Test Plan Section** or **Functional Coverage**.
- From the command line, use:

  coverage unlinked -r -plansection

  This displays all unlinked items.

Related Topics

- coverage unlinked command
- coverage tag command
- Finding Unlinked Items
- Links Between the Plan Section and the Coverage Item
- Finding Information for Missing Verification Plan Links
- Parameters Mapping Testplan Data Items

Finding Information for Missing Verification Plan Links

When you discover that you have an unlinked coverage item you want to link to the testplan, use the XML Import Hint to determine the exact path and name, as well as the type of that item. You can then copy this information directly to your verification plan.

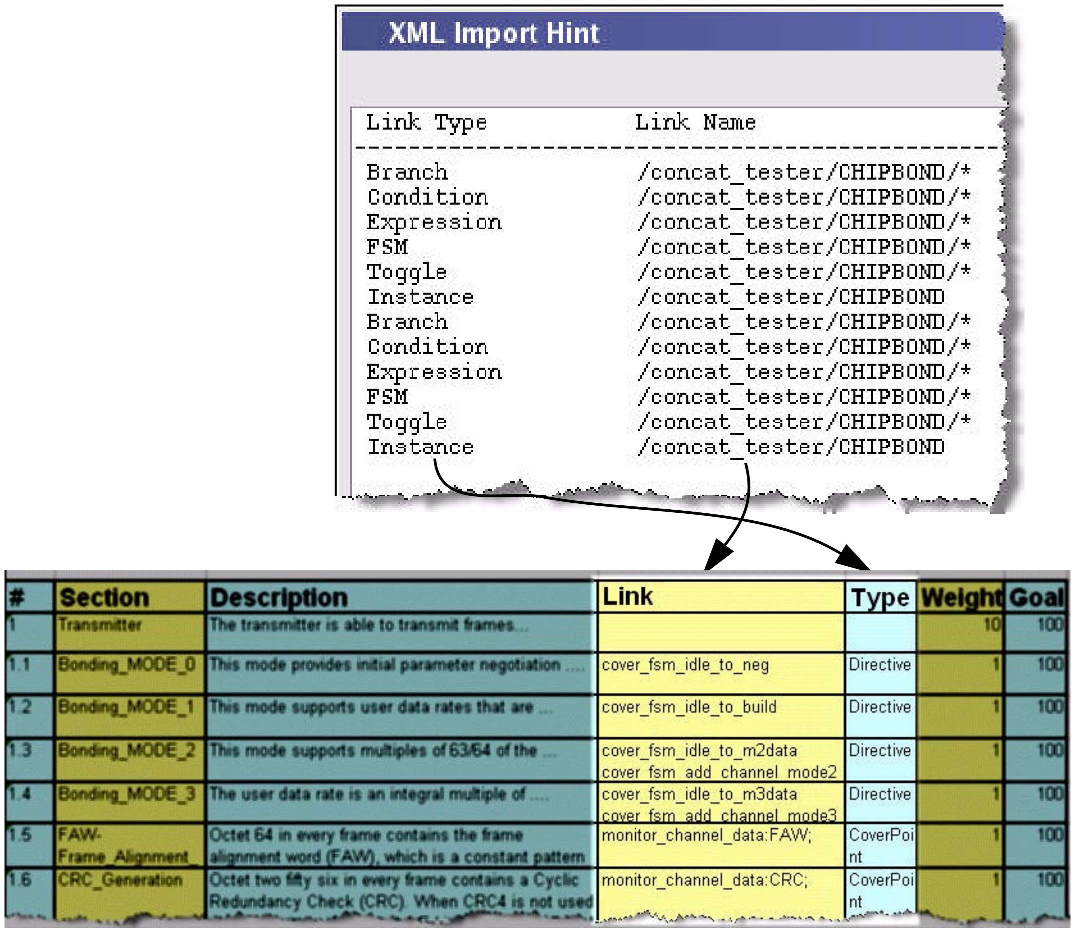
Procedure

1. In the Structure window, select the coverage item (design unit, instance, assertion, covergroup instance, or cover directive) to be linked.
2. Choose **XML Import Hint** from either of the following:
    - Right-click popup menu

- Dynamic menu pulldown (such as Instance Coverage or Assertions) in the main menu

The popup appears with the Link Type and the Link Name for the highlighted item, as shown in Figure 2-4. You can then enter this information into your verification plan and re-import it into a UCDB. When you next perform a merge with the plan, the link is complete.

Figure 2-4. Finding Link Information for Verification Plan



```
Link Type          Link Name
-----------------------------------------------
Branch             /concat_tester/CHIPBOND/*
Condition          /concat_tester/CHIPBOND/*
Expression         /concat_tester/CHIPBOND/*
FSM                /concat_tester/CHIPBOND/*
Toggle             /concat_tester/CHIPBOND/*
Instance           /concat_tester/CHIPBOND
Branch             /concat_tester/CHIPBOND/*
Condition          /concat_tester/CHIPBOND/*
Expression         /concat_tester/CHIPBOND/*
FSM                /concat_tester/CHIPBOND/*
Toggle             /concat_tester/CHIPBOND/*
Instance           /concat_tester/CHIPBOND
```

#	Section	Description	Link	Type	Weight	Goal
1	Transmitter	The transmitter is able to transmit frames...			10	100
1.1	Bonding_MODE_0	This mode provides initial parameter negotiation	cover_fsm_idle_to_neg	Directive	1	100
1.2	Bonding_MODE_1	This mode supports user data rates that are	cover_fsm_idle_to_build	Directive	1	100
1.3	Bonding_MODE_2	This mode supports multiples of 63/64 of the	cover_fsm_idle_to_m2data cover_fsm_add_channel_mode2	Directive	1	100
1.4	Bonding_MODE_3	The user data rate is an integral multiple of	cover_fsm_idle_to_m3data cover_fsm_add_channel_mode3	Directive	1	100
1.5	FAW-Frame_Alignment	Octet 64 in every frame contains the frame alignment word (FAW), which is a constant pattern	monitor_channel_data:FAW;	CoverPoint	1	100
1.6	CRC_Generation	Octet two fifty six in every frame contains a Cyclic Redundancy Check (CRC). When CRC4 is not used	monitor_channel_data:CRC;	CoverPoint	1	100

Related Topics

- coverage unlinked command
- Finding Unlinked Items
- Links Between the Plan Section and the Coverage Item
- Syntax for Links to the Coverage Model
- coverage tag command
- coverage analyze command
- Parameters Mapping Testplan Data Items

Link Debugging Using coverage tag Command

The "coverage tag" command can be used to explore design hierarchy by simply omitting the "-tagname" argument. Used in this way, it displays the tags (links) in a concise manner that shows the hierarchy of the design, which you would need in order to debug linking errors. With some knowledge of the designer's intent and the "coverage tag" command as a debug aid, you can figure out the intent for matching. You can then construct a particular set of entries in your testplan document to generate the corresponding tag command.

See coverage tag for syntax and an example.

Syntax for Links to the Coverage Model

Any coverage object within the UCDB can be linked to the testplan. The "coverage tag" command allows testplan scopes, coverage scopes, and bins within the database to be tagged. If a testplan scope, coverage scope or bin share the same tag— which is a string — then they are linked. Thus, coverage can be calculated and queried based upon the testplan. The xml2ucdb utility transforms information within the link, type and path values into the coverage tag syntax, which is then stored as an attribute of the testplan scope.

The following sections contain syntax and guidelines for the creation of these links:

Generic Link Entry Syntax Rules

The format used within the link, type and path values is the same — regardless of the document format used. In general, the following syntax for matching applies for all coverage types:

- Coverage items listed in the Link entry are space-delimited. Enter a space between each coverage item.
- If the string entered in a field begins with the path separator character ("/"), the string is assumed to be a hierarchical path to a specific design construct.
- Otherwise, the string is assumed to be a name (possibly including wildcard) which is used to search the design and link all design constructs of the specified type that match the pattern given.
- If the "Path" entry is a "-", the path is ignored.
- The "Path" entry may or may not be ignored, depending on the Type, as discussed in the relevant sections below.

Details for specific types of coverage, as well as any deviations from this link matching behavior, are detailed in the following individual Type sections.

- All-Purpose coverage tag Link
- Functional Coverage Links
- Code Coverage Links
- Bin Links
- Instances and Design Units Links
- Directed Test Links through Test Records
- Links for Classes
- XML Links

All-Purpose coverage tag Link

The "Tag" type — Allows you to link to any covergage item that can be tagged with the the "coverage tag" command. When Tag is specified as the Type, the coverage tag command (created by the xml2ucdb utility) uses only the command arguments specified in the "Link" column to create the link. This mechanism can be used to tag a greater variety of objects within the design, and takes advantage of the flexibility and variety of the linking capabilities of the coverage tag command itself.

"Link" Entry — See "coverage tag" for details.

“Path” Entry — Any information in this column is disregarded in favor of the information specified in the Link column.

Sample testplan section:

Table 2-8. Sample Testplan Section

Section	Title	Description	Link	Type
1.3	Branches_of_DU_work_.statemach	branches of DU work.statemach	“-du work.statemach -code b”	Tag

Functional Coverage Links

Functional coverage includes SystemVerilog covergroups, coverpoints and crosses, or bins; SystemVerilog "cover" statements; and PSL cover directives. This can also include SystemVerilog and PSL assertions. A link between the testplan and the functional coverage object is created by the name of the object within the source code. To make a link straightforward, you should ensure that the object to which something is being linked has a unique name within the environment. This is not always a straightforward task, so there are a number of mechanisms within the link format to provide greater flexibility.

Covergroups, Coverpoint and Crosses

This string defines the linked item as a covergroup, a coverpoint or a cross of a covergroup, and causes the coverage calculated for the complete covergroup, coverpoint or cross to be linked. There are a number of ways that the links can be defined to help ensure that each is unique.

Since SystemVerilog covergroups can be used a number of different ways in the HDL source, the formats for covergroups, coverpoints and crosses are flexible enough to take all the possibilities into consideration:

Table 2-9. Formats for covergroup, coverpoint, and cross Entries

Type	Link Formats
CoverGroup	<CoverGroupType>
CoverGroup	<CoverGroupType>,<CoverGroupInstanceName>;
CoverPoint	<CoverGroupType>:<CoverPointName>;
CoverPoint	<CoverGroupType>,<CoverGroupInstanceName>:<CoverPointName>;
Cross	<CoverGroupType>:<CrossName>;
Cross	<CoverGroupType>,<CoverGroupInstanceName>:<CrossName>;

- CoverGroupType — the type name given to the Covergroup within the SystemVerilog source code.
  - The simplest way to connect to a Covergroup type is to ensure that every type name used within the environment is unique.
  - If this is not possible, you can precede the covergroup type in all the forms shown above with a hierarchical path to the type, such as "/top/dut/".
  - If a path is required to make the Covergroup unique, you can enter this path information for a link within the Path value for a testplan scope.
- CoverGroupInstanceName — If the per_instance option has been used within the definition of the Covergroup, you can use the Covergroup instance name so that only the data from a single instance is tagged.

Tip: Take care when using instances of Covergroups in the naming: If the name of the instance is left up to the tool, it uses the name and the path of the variable that is used to construct the Covergroup. This leads to a complex string being used for the value of this parameter. A good rule of thumb is to use some mechanism to name the instance within the Covergroup with a unique string.

- CoverPointName — the name of the coverpoint within the Covergroup.
- CrossName — the name of the cross within the Covergroup.

“Path” entry requirements —

Any string specified is used to limit the search to a particular sub-hierarchy, unless full path is specified in the Link entry.

“Link” entry requirements —

- If the entry contains no commas or colons, the entered string matches according to the entered Type. The name of Type is case insensitive and the valid types are listed in Table 2-7.
- Coverage items listed in the Link entry are space-delimited. Enter a space between each coverage item.
- If the coverage item is a covergroup instance with an escaped identifier, the ‘instance’ field of the Link entry must start with a backslash. This must be followed by a trailing space, and — if the escaped identifier is the last thing in that linked item and another linked item follows in the same field— one more space (or the optional semicolon). Otherwise, the parser interprets the next linked item as a continuation of the string following the escaped identifier. The proper way to enter an escaped-identifier instance name into a link string is one of the following:

```
covergroup,\/cvg/instance/name ;
```

```
        covergroup,\/cvg/instance/name :coverpoint;
```

- If the Link entry begins with the path separator (/), the entry is interpreted as including the path to the item. The trailing path component is used as the Link string.
- If a path is specified in both Link and Path fields, the Link path takes precedent.

Assertions, Directives, and Generic Coverage Items

Items specified are linked to the full hierarchical path of the assertion, cover directive or other coverage item in the design. For any of these links to be formed, the following testplan entries must exist:

- Type entry — specified as “Assertion”, “Directive” or “CoverItem”

  A “CoverItem” can be used for any generic coverage item, and can be any hierarchical object — a covergroup or FSM coverage object; or a module instance or design unit. If it is a module instance or design unit, then all the coverage in that object is averaged together to compute the coverage inside that part of the testplan. A design unit will have only code coverage rolled up into it, not functional coverage or assertion data. See “Calculation of Total Coverage” for related details.
- Link Entry — Generic linking behavior. See “Syntax for Links to the Coverage Model”.
- Path Entry — Used to limit the search to a given sub-hierarchy in the design by specifying the top-most scope of that sub-hierarchy. Ignored if full path specified is in Link entry.

Code Coverage Links

It is possible to have a plan item linked with any coverage metric that is stored within the UCDB, including code coverage. Some examples of where this can be useful are:

- linking with the states or transitions within a state machine,
- monitoring toggles at interfaces, or
- including complete coverage numbers for instances or design units.

Branch, Condition, Expression, FSM and Toggle types link directly to the code coverage statistics for a particular design scope.

“Link” entry — Generic linking behavior. See “Syntax for Links to the Coverage Model”.

“Path” entry — Ignored for all code coverage types, regardless of Link entry.

Bin Links

- Type entry — Bin

- Link entry — Full hierarchical path to bin(s). See “Identifying the Full Path to a Bin”.

A bin item is linked within a testplan by specifying both the full hierarchical path of the bin in the Link column of the testplan, and the designation of “Bin” within the Type column of the testplan.

If you already have a UCDB containing bins you wish to link to, one excellent way to do it is using the Questa Excel Add-In. See “Questa Excel Add-In for Testplan Creation” for details on how to install the add-in, and “Adding Links with the Select Link Dialog” for details on how to link to bins.

Identifying the Full Path to a Bin

The full path to a bin can be attained most easily using the -bins switch to the coverage tag command. In order to run the coverage tag command, you must first:

1. Run your simulation.
2. Save the results to a UCDB.
3. Reopen the UCDB:

   vsim -viewcov <saved_UCDB>

4. To see all paths of the bins in the design tree, you could enter:

   coverage tag -bins -path / -recursive

5. To display all of the bins under a specific scope, 'concat_tester' for example, enter:

   coverage tag -bins -path /concat_tester/* -recursive

   A very small excerpt of the results that are displayed in the Transcript is the following:

```
...
# /concat_tester/monitor_registers/Bit4/auto['b0] [cvg bin] (no tags)
# /concat_tester/monitor_registers/Bit4/auto['b1] [cvg bin] (no tags)
# /concat_tester/monitor_registers/Bit5/auto['b0] [cvg bin] (no tags)
# /concat_tester/monitor_registers/Bit5/auto['b1] [cvg bin] (no tags)
...
```

6. Edit the testplan source (Excel spreadsheet, etc.) for each desired bin to be linked:
   a. Set the Type column to Bin.
   b. In the Link column, place only the first listed string listed in the Transcript window, which is the absolute path to that bin.
7. Re-run Import Testplan, or XML2UCDB command for the bin linking to take effect.

Bin Coverage

A bin is either considered covered or not covered. This is based on the LRM definition of “at least” as the value at which the bin moves from being uncovered to covered. Linking to a bin can only report a bin as either 0% or 100% covered, which yields little useful information. It's more useful to link to covergroups and crosses/coverpoints, as they can have coverage values anywhere between 0% and 100%, depending on the number of coverpoints and crosses or the number of bins covered, respectively.

You can override the value of AtLeast for individual testplan items. See Overriding at_least Values in Testplan for details.

Instances and Design Units Links

The strings you enter into the testplan for these items define either an instance name as a path or a design unit as a name as the link.

Instance and DU types link directly to the average coverage (code and functional) of the specified design instance or design unit.

“Link” entry — Specifies full path to the instance or design unit, whose aggregate coverage (the weighted average of all the coverage defined in the instance or the design unit) is to be linked to the testplan section. This includes all code, functional, assertion and user coverage metrics which are stored in the UCDB.

“Path” entry — Ignored.

Directed Test Links through Test Records

When the string “Test” is specified in the Type column/field, it allows a testname to be defined and linked to within the UCDB. It is also what allows the successful running of a directed test to be linked to the testplan. The successful running of the test case is defined by the “teststatus” attribute set in the UCDB for the particular “testname”. The value of the “teststatus” attribute is controlled automatically by the status of the simulation run. However, it can be overwritten with the CLI, based on any user defined function.

If the simulation is VHDL, then whatever result was the worst, immediate assertion severity level is what appears in the "teststatus" field (i.e. "Note", "Warning", "Error" or "Failure"). If desired, you can stop a “failure” or “error” severity from being set in VHDL by using the -ucdbteststatusmsgfilter argument with the vsim command. For example, let’s say you want to prevent the message “Done with Test Bench” from returning a “failure” or “error” teststatus. You can use the vsim command as follows:

```
vsim -ucdbteststatusmsgfilter "Done with Test Bench" -do stuff.do
```

or set the “UCDBTestStatusMessageFilter” *modelsim.ini* file variable:

```
UCDBTestStatusMessageFilter = "Done with Test Bench" "Ignore .* message"
```

If the simulation is SystemVerilog, then the worst result is reported with either $warning, $error or $fatal. The test coverage will show 100% if the testname is in the UCDB file and if the test has run successfully. In other words, if the test ran and passed, the coverage for that test attribute record is 100%. If it didn't run (or if it failed), the coverage is 0%. It is used to specify a set of directed tests that need to be run to achieve the testplan goal.

"Link" entry — The Link column should contain the value of the TESTNAME field for one or more test attribute records (see "Test Attribute Records in the UCDB"). The test name can contain wildcards to help specify one or more matches to a name. For example, if you enter "Test" as the Type, and "Error*" as the link, it will match one or more tests whose testname starts with "Error".

As with the other coverage types, if you have only one "Type" field containing the string "Test", you can have multiple space-separated test data record names in the "Link" field. The "Test" type can be mixed with other types in the same testplan section.

"Path" entry — Ignored.

Links for Classes

If a class based approach is taken to the environment, for example when using OVM or Mentor's Advanced Verification Methodology (AVM), it is still possible to pass instance names. In this case the Covergroup is defined in a class within a package, the hierarchical location needs to start from the package where the class definition is defined. The same rules for connection of the links should be followed as shown with the following example.

Example 2-10. Class-based Coverage Example

```
package AHB_VIP;
   `include avm_coverage_1.sv
endpackage

Inside avm_coverage_1.sv :
  class myavm_coverage extends avm_subscriber;

    covergroup ahb_cvg ( string p_name );
      option.per_instance = 1;
      option.name = p_name;
        ….
       coverpoint X
       cross Y
        ….
    endgroup

   function new(string nm = "", avm_named_component p)
     ahb_cvg=new(nm)
   endfunction
…..
endclass
```

Using the class :

```
local myavm_coverage ahb_cover;
ahb_cover    = new("ahb_inst1", this);
```

The Covergroup "ahb_cvg" is defined in a class called "myavm_coverage", within a package called "AHB_VIP". The class is constructed and the instance name "ahb_inst1" is passed to the constructor. In this case, the entries defined in Table 2-10 can be used to successfully link to the coverage within the class.

Table 2-10. Class Example Linking

Link	Type	Path
ahb_cvg,ahb_inst1;	CoverGroup	
ahb_cvg,ahb_inst1:X;	CoverPoint	
ahb_cvg,ahb_inst1:Y;	Cross	
/AHBVIP/avmCoverage/ahb_cvg,ahb_inst1;	CoverGroup	
ahb_cvg,ahb_inst1;	CoverGroup	/AHBVIP/avmCoverage
/AHBVIP/avmCoverage/ahb_cvg,ahb_inst1:X;	CoverPoint	
ahb_cvg,ahb_inst1:X;	CoverPoint	AHBVIP/avmCoverage
/AHBVIP/avmCoverage/ahb_cvg,ahb_inst1:Y;	Cross	
ahb_cvg,ahb_inst1:Y;	Cross	/AHBVIP/avmCoverage

XML Links

The "XML" type is used for nested testplan import. The "Link" field would be a partial command line (essentially, the command line for the nested import without the "xml2ucdb" or the UCDB file name). During import, the nested testplan sections are inserted into the section where the "XML" type is used.

If the "XML" type is used, it must be the only coverage item in the section. That section may not have any other coverage item links. Moreover, the entry in the "Link" field is parsed as a single string, so it’s not possible to nest more than one testplan in a single section.

“Path” entry — Ignored.

Related Topics

- coverage analyze command
- Links Between the Plan Section and the Coverage Item
- Syntax for Links to the Coverage Model
- Backlink Coverage Items to Plan with SV Attributes
- coverage unlinked command
- Finding Information for Missing Verification Plan Links
- Finding Unlinked Items

Verification Tracker Window

The Verification Tracker window is used for test traceability analysis. The functionality is also available through the command line with the coverage analyze command.

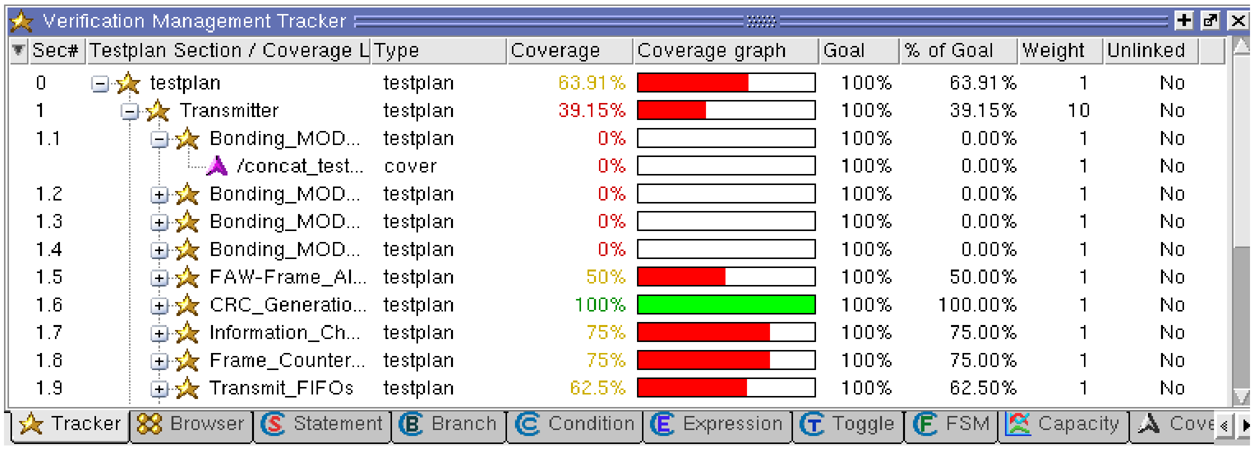
For details on how the Total Coverage column statistics are calculated, see "Calculation of Total Coverage" for coverage statistics calculation details.

Accessing

Access the window using:

- Menu item: **View** > **Verification Management** > **Tracker**
- Command: view testtracker

Figure 2-5. Verification Tracker Window



Sec#	Testplan Section / Coverage L	Type	Coverage	Coverage graph	Goal	% of Goal	Weight	Unlinked
0	testplan	testplan	63.91%		100%	63.91%	1	No
1	Transmitter	testplan	39.15%		100%	39.15%	10	No
1.1	Bonding_MOD...	testplan	0%		100%	0.00%	1	No
	/concat_test...	cover	0%		100%	0.00%	1	No
1.2	Bonding_MOD...	testplan	0%		100%	0.00%	1	No
1.3	Bonding_MOD...	testplan	0%		100%	0.00%	1	No
1.4	Bonding_MOD...	testplan	0%		100%	0.00%	1	No
1.5	FAW-Frame_Al...	testplan	50%		100%	50.00%	1	No
1.6	CRC_Generatio...	testplan	100%		100%	100.00%	1	No
1.7	Information_Ch...	testplan	75%		100%	75.00%	1	No
1.8	Frame_Counter...	testplan	75%		100%	75.00%	1	No
1.9	Transmit_FIFOs	testplan	62.5%		100%	62.50%	1	No

By default, the Testplan column layout is used. However, you can customize the columns that appear in the Tracker window:

- Right-click in the column headings to display a list of all column headings
- Toggle the columns on or off.

You can also create custom column layouts which will preserve the columns you commonly use. Refer to "Viewing and Analyzing Verification Data" for information on using this window to analyze your merged testplan results.

Viewing Associated Plan and Test Data

In order to view data in this window, you must:

- use a merged file (UCDB) containing testplan and test results, with testplan items linked to coverage
- the file must be loaded with vsim -viewcov

Verification plan items can be linked to coverage items either automatically or manually:

- automatically — during merge if the testplan UCDB was generated by either the Test Plan Import facility or the xml2ucdb command, or
- manually — using the coverage tag command

For more information, see "Combining Verification Plan Data and Test Data."

Saving Verification Tracker Column and Filter Settings

Save your column layout and any filter settings to an external file (*tracker_column_layout.do*) by selecting **File** > **Export** > **Column Layout** while the window is active. You can reload these settings with the do command. This export does not retain changes to column width.

Viewing Verification Tracker Data in HTML

You can export the view of the columns and verification data in the Tracker, including any filter settings, to an HTML file (*tracker.htm*) using the by selecting **File** > **Export** > **HTML** while the window is active. A dialog opens in which you can set both the name of the file and the directory to which it is saved. By default, the HTML file is saved into the current directory.

GUI Elements of the Verification Tracker Window

This section provides an overview of the GUI elements specific to this window.

Column Descriptions

Table 2-11. Verification Management Tracker Window Column Descriptions

Column Title	Description
% of Goal	lists the percentage of goal reached by the section of plan. By default, this number is calculated hierarchically: • For parent testplans without a Goal, this number is defined as weighted average of child "% of Goal" values. Any linked coverage items in the parent testplan scope are included in that calculation, where all the coverage items are considered together as a single entity for contributing to the parent testplan's "% of Goal". • For testplans with a Goal (either parent or leaf), "% of Goal" is simply Coverage / Goal.
Coverage	lists coverage percentage for that section of plan
Coverage graph	graphical illustration of percentage of coverage
Description	a description of section in plan
Goal	lists the goal percentage for that section in plan; missing Goal value is displayed as "-". Double-click on value to enter a new value. When you explicitly override the Goal of a testplan, "% of Goal" is defined as Coverage / Goal. Results from children are ignored.
Sec#	# of the section in plan
Type	the type for that section, from the Type column in plan
Unlinked	list whether item has link from coverage item to testplan section
Weight	lists the weight assigned. Double-click on value to enter a new value.

Menu Items

The following menu items are related to the Verification Tracker window, and are based on the coverage analyze command.

- **View Coverage Details** — valid when any covergroup, coverpoint, cross, assertion and or cover directive is selected in the testplan. Opens up a new tab of that item's coverage type, highlighting the coverage item, where further coverage details can be viewed.
- **Configure Selected** — displays the Coverage Configurations Dialog Box, which allows you to set the goal and weight for the specified test(s).
- **Test Analysis** — analyzes the results to:
  - Find Tests with Least Coverage
  - Find Tests with Most Coverage

    - Find Tests with Zero Coverage
    - Find Tests with Non-Zero Coverage
    - Rank Most Effective Test - Opens a dialog where you can select ranking criteria to apply.

  When you select a section in the testplan and apply the above menu items, a new window opens displaying the tests with the least, most, zero, etc. coverage.

- **Find Unlinked** — reports on testplan sections that are unlinked with coverage; or, conversely, design or coverage objects which have not been linked with a testplan. See "Finding Unlinked Items".
    - Testplan Section — finds unlinked testplan sections.
    - Functional Coverage — finds unlinked functional coverage items.
- **Filter** — either opens the Filter Setup Dialog Box, or applies desired filter setups.
    - Setup — opens the Filter Setup dialog that allows you to save and edit filters to apply to the data.
        - Create button — opens the Create Filter dialog which allows you to select filtering criteria, and select the tests for application of the specified filters. When you enter a Filter Name, and select "Add", the Add/Modify Selection Criteria dialog box is displayed, where you can select the actual criteria to filter. See "Filtering Results by User Attributes" for an example.
    - Apply — applies the selected filter(s) on the data.
- **Summary** — displays the Tracker Summary Dialog Box, which allows you to create a summary report based on the selected UCDB files. Corresponds to the coverage analyze command's -summary argument.
- **Re-import Testplan and Refresh** — re-imports selected verification plan and refreshes the contents. See "Refreshing Tracker after Changing a Plan" for more details.
- **Set Precision** — allows you to control the decimal point precision of the data in the Verification Browser window.
- **Configure Colorization** — opens the Colorization Threshold dialog box which allows you to control the colorization of coverage results displayed in the "Coverage" column, as well as set the low and high threshold coverage values for highlighting coverage values:
    - < low threshold — RED
    - > high threshold — GREEN
    - > low and < high — YELLOW
- **Expand / Collapse Selected** — Expand or collapse selected UCDBs.

- **Expand / Collapse All** — Expand or collapse all UCDBs.

Verification Test Analysis Window

The Verification Test Analysis window is used to perform several test and coverage analysis query functions. The functionality is also available through the command line with the coverage analyze command.

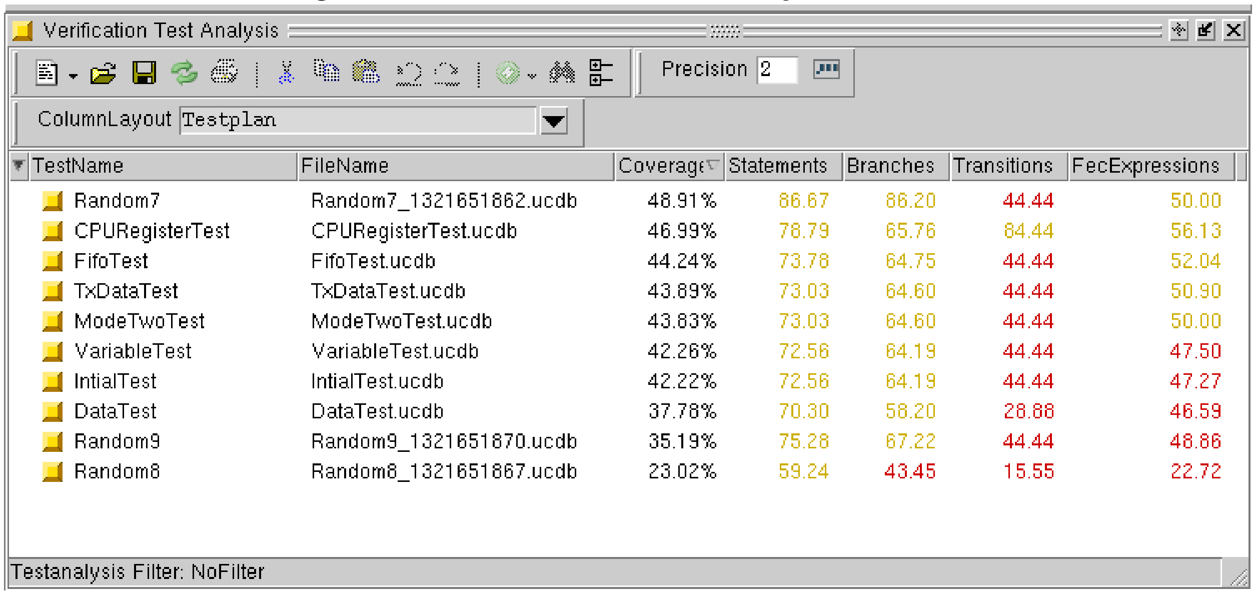
For details on how the Total Coverage column statistics are calculated, see "Calculation of Total Coverage" for coverage statistics calculation details.

Accessing

Access the Test Analysis window using:

- Menu item: in the **Verification Tracker** window or **Covergroups** window, right click on any item in window, and select any of the **Test Analysis** > menu choices:
    - Find Tests with Least Coverage
    - Find Tests with Most Coverage
    - Find Tests with Zero Coverage
    - Find Tests with Non-Zero Coverage
    - Rank Most Effective Tests...
    - Summary...
- Menu item: in the **Assertions** or **Cover Directives** windows, right click on any item in window, and select any of the **Test Analysis** > menu choices:
    - Find Tests that Hit
    - Find Tests that Miss
    - Summary...
- Command: view test analysis

Figure 2-6. Verification Test Analysis Window



TestName	FileName	Coverage	Statements	Branches	Transitions	FecExpressions
Random7	Random7_1321651862.ucdb	48.91%	86.67	86.20	44.44	50.00
CPURegisterTest	CPURegisterTest.ucdb	46.99%	78.79	65.76	64.44	56.13
FifoTest	FifoTest.ucdb	44.24%	73.78	64.75	44.44	52.04
TxDataTest	TxDataTest.ucdb	43.89%	73.03	64.60	44.44	50.90
ModeTwoTest	ModeTwoTest.ucdb	43.83%	73.03	64.60	44.44	50.00
VariableTest	VariableTest.ucdb	42.26%	72.56	64.19	44.44	47.50
IntialTest	IntialTest.ucdb	42.22%	72.56	64.19	44.44	47.27
DataTest	DataTest.ucdb	37.78%	70.30	58.20	28.88	46.59
Random9	Random9_1321651870.ucdb	35.19%	75.28	67.22	44.44	48.86
Random8	Random8_1321651867.ucdb	23.02%	59.24	43.45	15.55	22.72

Testanalysis Filter: NoFilter

GUI Elements of the Verification Test Analysis Window

This section provides an overview of the GUI elements specific to this window.

Column Descriptions

The columns that appear in the Test Analysis window are the same as those in the Verification Management Browser window. They can be either added to the view or removed from view by selecting the Configure Column dropdown icon to the left of the first column name.

Data contained in the columns is can be filtered using RMB > Filter > Setup/Apply. For a description of how to set and apply filters, see “Filtering Results by User Attributes”.

Menu Items

The following menu items are related to the Verification Test Analysis window, and are based on the coverage analyze command.

- **Merge** — displays the Merge Files Dialog Box, which allows you to merge any selected UCDB files. Refer to the section Merging Coverage Data for more information.
- **Rank** — displays the Rank Files Dialog Box, which allows you to create a ranking results file based on the selected UCDB files. Refer to the section Ranking Coverage Test Data for more information.
- **HTML Report** — displays the HTML Coverage Report Dialog Box, which allows you to view your coverage statistics in an HTML viewer.
- **Command Execution** — allows you to re-run simulations based on the resultant UCDB file based on the simulation settings to create the file. You can rerun any test whose test

record appears in an individual *.ucdb* file, a merged *.ucdb* file, or ranking results (*.rank*) file. See "Test Attribute Records in the UCDB" for more information on test records.

- Setup — Displays the Command Setup Dialog box, which allows you to create and edit your own setups which can be used to control the execution of commands. "Restore All Defaults" removes any changes you make to the list of setups and the associated commands.
- Execute on all — Executes the specified command(s) on all *.ucdb* files in the browser (through **TestReRun**), even those used in merged *.ucdb* files and *.rank* files.
- Execute on selected — Executes the specified command(s) on the selected *.ucdb* file(s) through **TestReRun**.

- **Filter** — either opens the Filter Setup Dialog Box, or applies desired filter setups.
    - Setup — opens the Filter Setup dialog that allows you to save and edit filters to apply to the data.
        - Create button — opens the Create Filter dialog which allows you to select filtering criteria, and select the tests for application of the specified filters. When you enter a Filter Name, and select "Add", the Add/Modify Selection Criteria dialog box is displayed, where you can select the actual criteria to filter. See "Filtering Results by User Attributes" for an example.
    - Apply — applies the selected filter(s) on the data.
- **Generate Vrun Config** — generates Verification Run Management configuration file (*.rmdb*) including selected tests or all tests in the directory. Selecting either option brings up a dialog to enter the name to be used for the *.rmdb* file.
    - Save Selected Tests — Saves selected tests into a *.rmdb* file to be executed by vrun command.
    - Save All Tests — Saves all tests in the directory into a *.rmdb* file to be executed by vrun command.
- **Show Full Path** — toggles whether the FileName column shows only the filename or its full path.
- **Set Precision** — allows you to control the decimal point precision of the data in the Verification Browser window.
- **Configure Colorization** — opens the Colorization Threshold dialog box which allows you to off the colorization of coverage results displayed in the "Coverage" column, as well as set the low and high threshold coverage values for highlighting coverage values:
    - < low threshold — RED
    - > high threshold — GREEN

    - > low and < high — YELLOW

Chapter 3 Questa Excel Add-In for Testplan Creation

The Questa Testplan Tracking Excel Add-In is an Excel COM Add-In which aids Questa users in the creation of testplans for Questa Verification Management.The add-in provides assistance with the creation and capture of the testplan spreadsheet, which is used as the front-end to Questa's testplan tracking tool. The functionality provided can be broken down to the following areas:

- Generating the testplan's structure
- Guiding the linking of the coverage model
- Exporting data to Questa and the UCDB
- Annotating coverage data back into the spreadsheet

For information and instructions on the tasks associated with these areas of functionality, see "Using the Questa Add-In".

For reference information on the Excel Add-In GUI, see "GUI Reference for the Questa Add-In".

For information on how, when and why to save testplans either to XML or UCDB, see "Saving and Exporting to XML and UCDB".

Installing the Excel Add-In

The self extracting installer, *QuestaExcelAddIn.msi*, can be found in the *$MTI_HOME/vm_src* directory for both Unix and Windows versions of Questa.

The Questa Excel Add-In works only on Windows. Unix users can copy the installer to a Windows machine and install the Add-In.

UCDB files are platform independent, therefore UCDBs generated on Linux/Unix can be read on the Windows platform.

Prerequisites

Ensure that you have access to the Questa SIM software on your Windows machine. Use of the majority of the Excel Questa Add-In's features requires access to two applications which are part of the standard Questa SIM installation: the xml2ucdb program, and a UCDB API application. This provides the spreadsheet with the ability to interface directly with UCDB files.

Regardless of whether you have already installed Questa on your Windows machine, before you can install the Excel Add-In, you must verify that your machine has:

- Microsoft Excel 2003, 2007, or 2010.
- Microsoft .NET Framework version 2 or higher for Windows XP. By default, Microsoft Windows Vista and Microsoft Windows 7 have a higher version of .NET Framework and Windows Installer installed.

Installing the Add-In

Installing the Questa Excel Add-In is simple. The steps are:

For Windows Vista: Consult the Prerequisite section for necessary preliminary steps.

1. Run the installer:

   $MTI_HOME/vm_src/QuestaExcelAddIn.msi

2. Follow the step by step instructions which appear in the Setup Wizard.

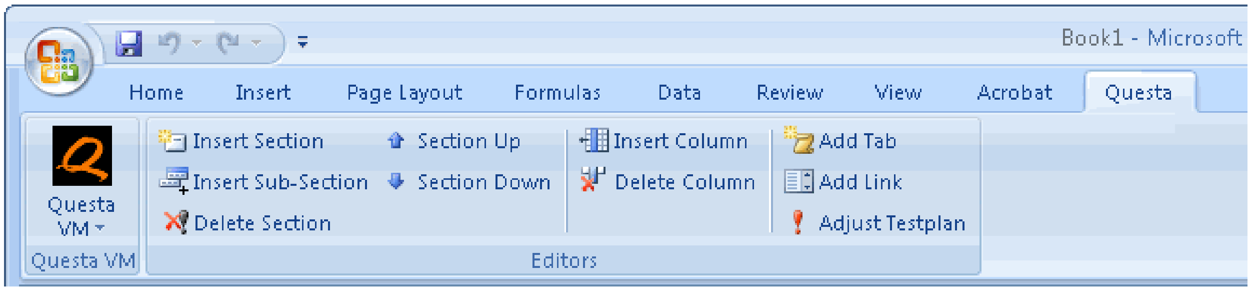
The Installer adds all necessary files for the Add-In functionality and makes the necessary registry changes to integrate the Add-In with Excel.

Validation

If the Add-In has been successfully installed, depending on which version of Excel you are using, the following is observed.

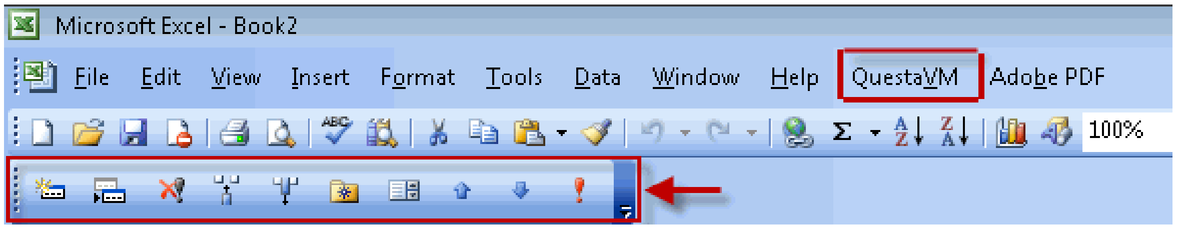
In Excel 2007/2010: a new tab, called **Questa**, is added to the Excel ribbon. The tab includes two groups: QuestaVm and Editors Group, as shown below.

Figure 3-1. Questa Tab in Excel 2007 or 2010



In Excel 2003: a new menu is added to the Worksheet Menu Bar, called QuestaVM. Also, A new tool bar (Questa Toolbar) is added, as shown below.

Figure 3-2. Questa Toolbar in Excel 2003



If you do not see the Questa menu/tab, consult the following document for possible solutions:

$MTI_HOME/vm_src/AfterInstallReadMe.txt

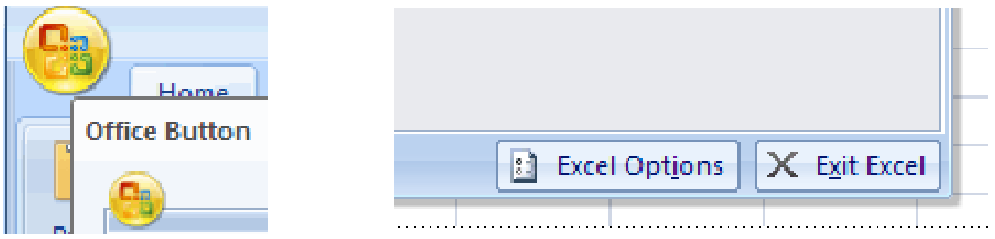
Disabling the Questa Excel Add-In

If you want to disable the Add-In within Excel, without removing the installation, follow the instructions in this section.

Excel 2007 or 2010:

1. Open Microsoft Excel 2007 or 2010 and click the Office Button.
2. Select Excel Options.

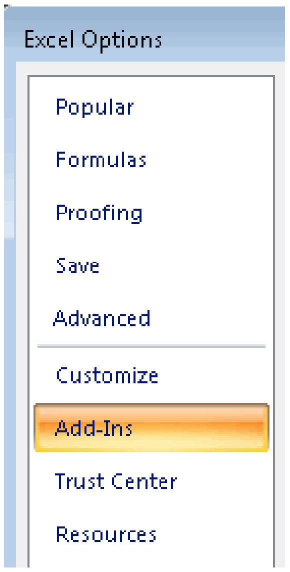
Figure 3-3. Office button and Excel Options



The Excel Options dialog box opens with a menu of options.

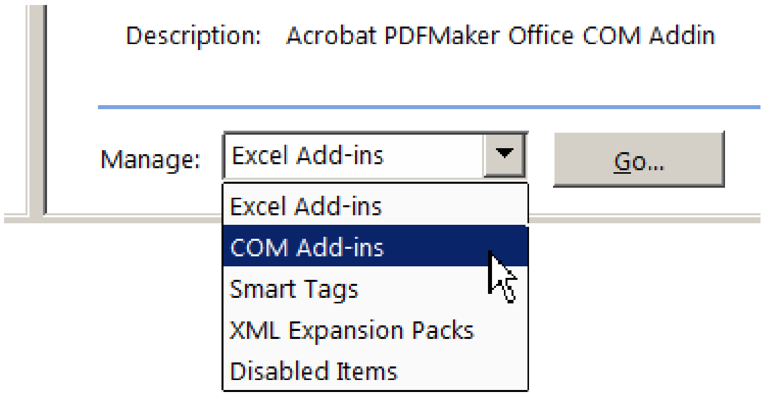
3. Select Add-Ins from the options menu.

Figure 3-4. Excel Options menu



4. At the bottom of the dialog, in the **Manage:** drop down box, choose **COM Add-ins** and select “**Go…**”.

Figure 3-5. Excel 2007/2010 Manage dropdown menu


Description: Acrobat PDFMaker Office COM Addin

This opens a dialog entitled COM Add-Ins.

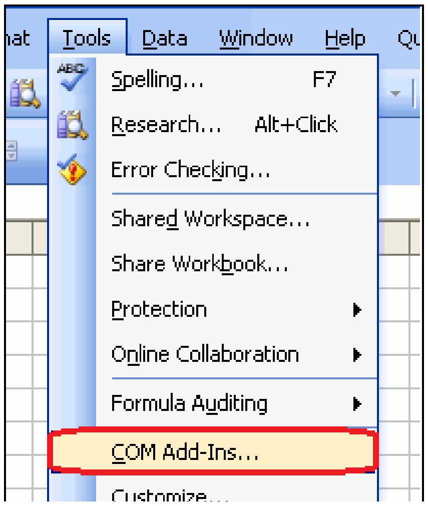
5. Un-check the box marked **Questa Excel Add-in** and select **OK**.

   This disables the Questa Excel Add-In, removing the Questa tab from the ribbon bar.

Excel 2003:

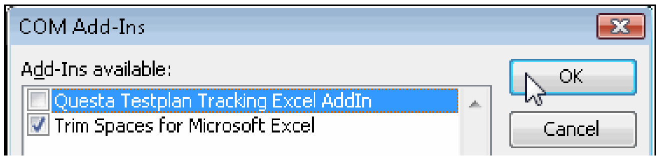
1. From the Tools menu, select **COM Add-Ins**. This opens the COM Add-In dialog box.

Figure 3-6. Excel 2003 Tools Menu



2. In the COM Add-In dialog, uncheck the **Questa Testplan Tracking Excel Add In**.

Figure 3-7. COM Add-In dialog



☐ Questa Testplan Tracking Excel AddIn
☑ Trim Spaces for Microsoft Excel

GUI Reference for the Questa Add-In

The Questa Excel Add-In is split into two sets of functionality: one set is accessed through the Questa VM menu, the other set through the Questa Toolbar.

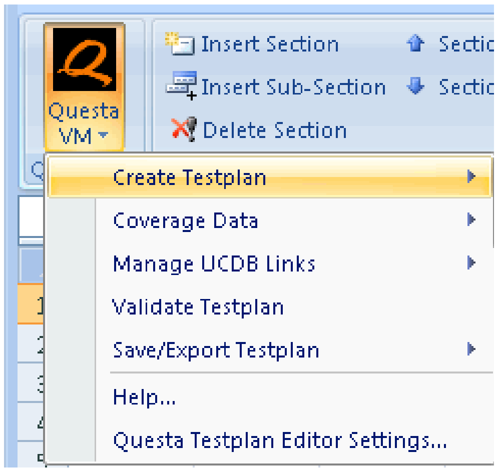
- Use the Questa VM Menu to:
  - Create testplans — either from scratch or from an existing UCDB
  - Annotate or remove coverage in the testplan from a UCDB
  - Create or remove links from testplan to UCDB
  - Validate the syntax of your testplan
  - Save/Export testplans— to XML or to a UCDB
- Use the Questa Toolbar to:
  - Add/Delete testplan sections or sub-sections to the plan
  - Move sections up or down in the testplan
  - Insert/remove testplan columns

- Add links to testplan
- Add sheets (tabs) to testplan

The Questa VM Menu

The Questa VM menu is activated when the Questa tab is selected within the Excel ribbon. The functions that can be run from the Questa VM menu are detailed in this section.

Figure 3-8. Questa VM Menu for Excel Add-In



Click on the Questa VM icon to display the popup menu and select one of the following options:

Table 3-1. Questa VM Popup Menu

Popup Menu Item	Menu Selections and Descriptions
Create Testplan	New — Opens a dialog for entering Testplan Name (or Sheet Name) and alternating colors for the testplan's columns.
	From Testplan UCDB — Allows you to select a UCDB with a testplan which has already been imported, and to set both the spreadsheet name that is generated within Excel, as well as the alternating colors for the columns.
Coverage Data	Annotate — Reads the coverage data from a UCDB for the testplan and annotates it into the selected spreadsheet for columns whose title names are "coverage" and "objects". Then, the coverage number for each section is annotated into a column with the name "Coverage"; the number of objects linked for that section is annotated into a column named "Objects". The totals for both coverage and objects are also annotated, and appear at the top of the spreadsheet. Successful annotation depends on section numbers in the spreadsheet matching those in the UCDB.
	Remove — removes all annotated coverage from a spreadsheet.

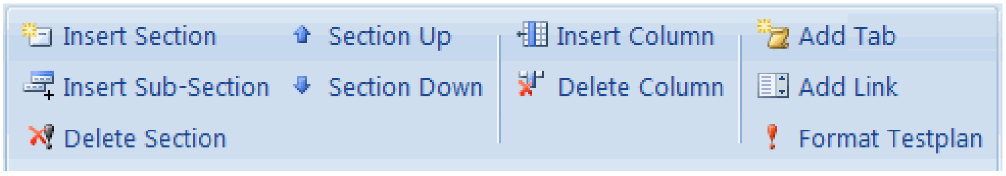
Table 3-1. Questa VM Popup Menu (cont.)

Popup Menu Item	Menu Selections and Descriptions
Manage UCDB Links	Create — create or change an existing link (association) between a UCDB and a spreadsheet, for the purposes of linking. This is automatically run when the "Add Link" tool bar button is selected.
	Remove — removes the association between UCDB and spreadsheet. If a different UCDB file needs to be used for the linking process then the association with a current UCDB file must be broken. This function deletes the old cache link files so that a new association with a different UCDB file can be set when using the link dialog (see Add Link in the "The Questa Toolbar").
Validate Testplan	Checks syntax and format within your spreadsheet. The validation is performed in interactive or non-interactive mode, which you select using the Validation Type (see "Settings Dialog"). For a list of validation checks performed, see "How and Why to Export Testplan to UCDB".
Save/Export Testplan	Save and export it so that it can be imported into Questa. The Excel spreadsheet testplan can be saved and stored in any format that Excel supports. It can then be exported to XML, or directly to a UCDB. Note: Carefully select the Save As Type selection for UCDB compatibility. See "How and Why to Export Testplan to UCDB" for details on this, and why you might choose one format over the other.
Help...	Opens this documentation in a PDF file.
Questa Testplan Editor Settings...	Opens the Settings dialog for setting: the type of validations run, the type of coverage being imported, coverage items for unimplemented links, path to <xml2ucdb>.ini file and other configuration settings.

The Questa Toolbar

The Questa toolbar has seven buttons and can be opened by selecting the Questa tab in the Excel ribbon.

Figure 3-9. Questa Toolbar: Features and Icons



- Insert Section — Inserts a new plan section. Select the row at which you want to insert a new plan section and hit this button. This adds a new section at the same numbering level as the selected section. It picks the next level number and then scan down the rows of the spreadsheet to re-number the other section numbers.

- Insert Sub-Section — Inserts a new sub plan section. Select the row at which you want to insert a new sub-section and hit this button. This adds a sub-section of the selected numbered section. In other words, if you select section 1, it adds section 1.1; if you select section 1.2, it adds section 1.2.1.
- Delete Section — Removes a section from the spreadsheet. Select the row to remove from the plan, and hit this button. Children must be removed before you can remove a parent section of the testplan. The sections are renumbered after the a section's removal.
- Section Up — Moves the selected section upwards in its hierarchy. The previous section will move downwards. If the selected section is the top section in its hierarchy, an error pop up message will show up.
- Section Down — Moves the selected section downwards in its hierarchy. The following section will move downwards. If the selected section is the last section in its hierarchy, an error pop up message will show up.
- Insert Column — Adds a column attribute to the spreadsheet. Questa supports any number of user columns to carry information about the verification plan items. You are able to add or remove any one of the user attributes columns.

  To add a column, select the column that you would like the new attribute/column to be added and hit the button. This displays the pop-up dialog that requests a name for the column. Once the name is entered and the "OK" button is selected, the column is added to the spreadsheet. The alternating color of the columns within the spreadsheet is automatically adjusted.
- Delete Column — Removes the selected column. Adjusts the alternating color of the columns, if necessary.
- Add Tab — Adds a tab to split the testplan up across multiple tabs in the workbook. Hitting this button adds a new tab, copying over the format of the testplan from the current sheet, using a testplan section that starts at a section number one higher than the highest parent section number in the other tabs in the workbook.
- Add Link — Brings up the Select Link dialog to facilitate linking of coverage data to the spreadsheet.

  The link entry makes a connection between the spreadsheet and a UCDB file. Coverage object information is cached into a workbook. Any changes to the date stamp of the UCDB causes the data in this cache files to be automatically updated. It you want to use a different UCDB file, you must remove the association between the UCDB file and the spreadsheet by selecting the **Manage UCDB Links > Remove** menu item, which deletes the cache workbook. See "Managing Links".
- Format Test Plan — Opens the Format Testplan dialog box for formatting an existing testplans that may contain empty rows within sections. When clicked, a dialog appears

asking for a definition of the location of the testplan within the active spreadsheet. To determine the boundaries of the actual testplan to be created, it asks you to enter the letter of Excel columns where the valid information starts and ends, as well as the starting and ending (last) row in the testplan. When you select **OK**, it fills any empty cells in the section number column with '!'. Rows with column numbers which contain "!" are interpreted as comment rows, and are ignored during any operation.

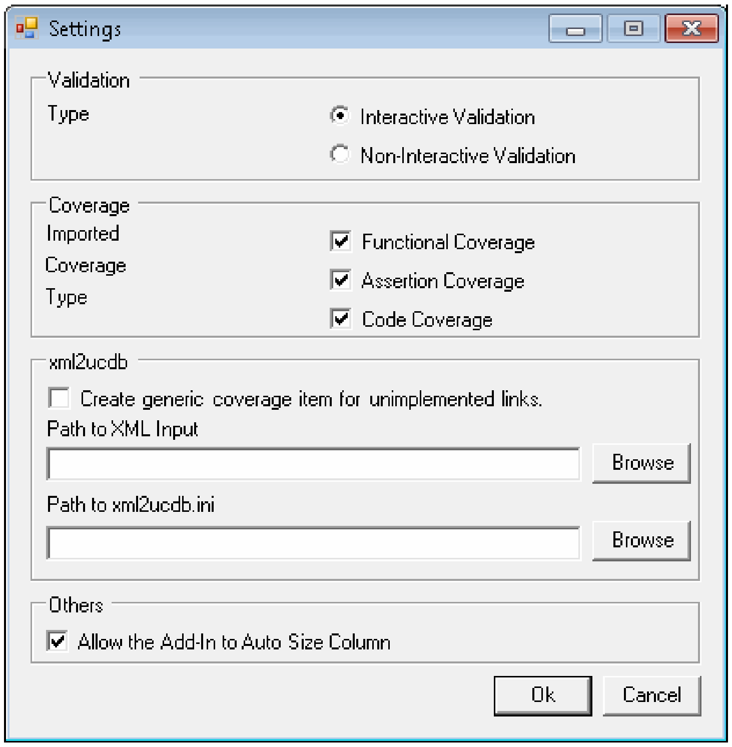
Settings Dialog

To access: Excel's Questa Tab > Questa VM > Questa Testplan Editor Settings

Description

Use this dialog to control the type of validation and the types of coverage to be imported from the UCDB. Once selected, the settings are remembered by the Add-In.

Figure 3-10. Settings Dialog


Settings

Validation
Type: (● Interactive Validation) ○ Non-Interactive Validation

Coverage
Imported Coverage Type: ☑ Functional Coverage ☑ Assertion Coverage ☑ Code Coverage

xml2ucdb
☐ Create generic coverage item for unimplemented links.
Path to XML Input: [ ] Browse
Path to xml2ucdb.ini: [ ] Browse

Others
☑ Allow the Add-In to Auto Size Column

Validation Type:

- Interactive Validation — guides you to the row containing a problem, where you fix the problem and run the validation again, stepping through all the problems one by one, until all problems are resolved.
- Non-Interactive Validation — the entire spreadsheet is validated and a report containing a complete list of all problems opens in a window for viewing.

Imported Coverage Type:

- Functional Coverage, which includes covergroups, coverpoints, crosses and directives
- Assertion coverage, which includes assertion data
- Code Coverage, which includes statement, branch, expression, condition, toggle and FSM coverage.

xml2ucdb — These settings control the UCDB export options (see the xml2ucdb command for further details):

- Create generic coverage item for unimplemented links — creates a single bin for each unlinked item (a link in the testplan which is not found in the design) when either the Unimplemented column is not present in the testplan, or the corresponding link doesn't have any entry in the Unimplemented column.
- Path to XML Input — correlates to the -searchpath argument to xml2ucdb where you specify the XML file to use as a nested testplan. See "About Hierarchical Testplans" in the Questa SIM Verification Management User's Manual.

The settings in the Others area of the settings dialog includes the **Add-In to Auto Size Columns**. When checked, the Add-In re sizes the columns of the testplan to automatically fit Data in the column.

Related Topics

How and Why to Export Testplan to UCDB
Counting Coverage Contribution from Unimplemented Links
xml2ucdb command

Using the Questa Add-In

The following section is a quick start guide to using the plug-in; it assumes that the Add-In has already been installed. There are three starting points to using the Add-In once it is installed. You can either:

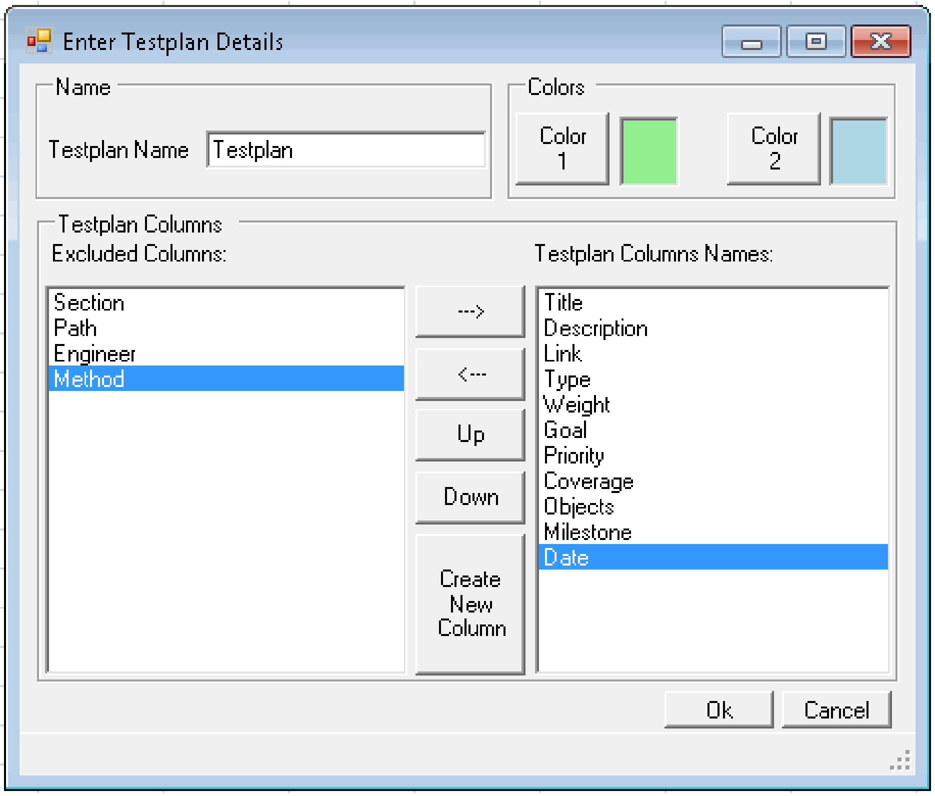
- create a new testplan from scratch using the creation dialog,
- generate a testplan spreadsheet from the testplan information stored in a UCDB, or
- use an existing Excel spreadsheet testplan which has already been formatted to be used within Questa testplan tracking.

You can also use the Add-in to manage the "linkage" of a spreadsheet to a particular UCDB, as well as the individual links between testplan sections and coverage items.

Creating a New Testplan from Scratch

The flow for creating a new testplan from scratch is as follows:

1. Start Excel and go to the Questa VM menu in the Add-In, select **Create Testplan** > **New** to display the creation dialog below.



Testplan Name: Testplan

Color 1: 🟩
Color 2: 🟦

Excluded Columns:
Section
Path
Engineer
Method

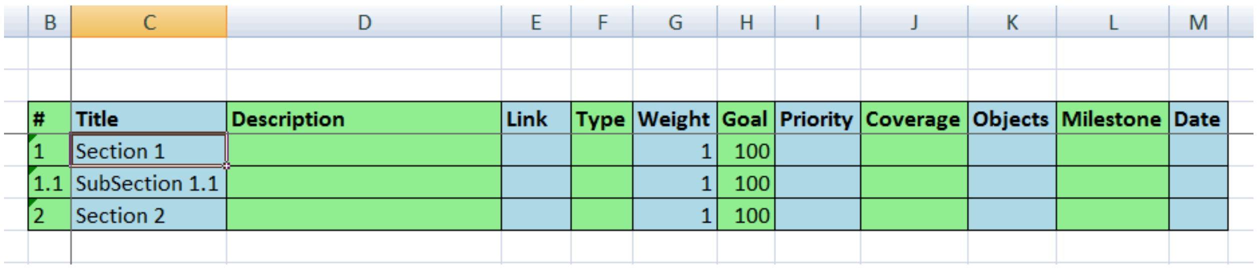
Testplan Columns Names:
Title
Description
Link
Type
Weight
Goal
Priority
Coverage
Objects
Milestone
Date

2. Use the left and right arrow buttons in the middle of the dialog to include or exclude the columns in the testplan you are creating. By default, the dialog opens with the default set of columns required by the testplan in a list on the right side of the dialog.

3. Order the columns as you want them to appear in the testplan using the “Up” and “Down” buttons.

4. Use the create button to create any attribute that you wish to track, for example “Milestones” or “Dates”, and have them added and ordered in the right hand list.

5. Hit the **OK** button and the new spreadsheet is generated using your selections, shown below.



#	Title	Description	Link	Type	Weight	Goal	Priority	Coverage	Objects	Milestone	Date
1	Section 1				1	100					
1.1	SubSection 1.1				1	100					
2	Section 2				1	100					

6. Add and Remove Sections and Column Attributes.

   Once the spreadsheet is created, you can use the toolbar to add and remove testplan sections and column attributes. For example, you can select section "1.1" and hit the **Insert Section** icon. This inserts a new section "1.2" into the spreadsheet and performs any necessary re-numbering. If you want to add a sub-section, select a section (for example "1.1") and hit the "Insert Sub-section" icon. This adds sub-section "1.1.1".

   Use the **Delete Section** to remove sections. The spreadsheet re-numbers the sections after deletion.

7. Insert and Delete Columns in the spreadsheet.

   To insert a column:

   a. Select a column where you want the new attribute to be placed and select the "Insert Column" icon.

      This opens a dialog to name the column and then insert the column with the new name and adjusts the colors within the columns to the right of the inserted column.

   b. Selecting the "Delete Column" icon removes the attribute and adjusts the colors to the right of the removed column.

8. Adding Links from the spreadsheet to a UCDB.

   One of the most useful features of the Add-In is the ability to interface with a UCDB file to fill in the Link and Type columns of the spreadsheet.

Note

If your current spreadsheet is already linked to a particular testplan other than the one you want to be linked with, you must remove that association (link) before you can select another UCDB for linking. Remove the association by selecting: **VM menu** > **Manage UCDB Links** > **Remove**.

   For detailed instruction on adding links, see "Adding Links with the Select Link Dialog"

9. Validate your testplan.

Once you have completed your link information and before outputting the XML file to take into Questa or generating the UCDB file, use the validate command to validate the format of your testplan. Select **Validate Testplan** from the Questa VM menu. This set can be run interactive or non-interactive depending on the setting within the settings dialog (see "Settings Dialog" and "How and Why to Export Testplan to UCDB").

10. Export the testplan for use with Questa. The spreadsheet testplan itself can be saved in any format that Excel supports. The Add-In has its own export function to generate files for use with Questa. You can chose to either export an XML file that can be used as the input to xml2ucdb using **Save/Export Testplan > to XML**; or you can write a UCDB file with the testplan information directly from the spreadsheet using **Save/Export Testplan > to UCDB**. See "How and Why to Export Testplan to UCDB" for details regarding this process.

Generating a Testplan from a Testplan UCDB

It is possible to have a testplan spreadsheet generated from a UCDB file which already contains testplan information. This could be a UCDB file that has been generated from another testplan via xml2ucdb, therefore could have come from another spreadsheet format, a word document or any other document format. This is useful if you want the spreadsheet to be in a common format generated by the Add-In, and it has the added advantage of knowing that all the functions of the Add-In are compatible with the generated format.

To generate a testplan from a testplan UCDB:

1. Select the **Create Testplan > from Testplan UCDB** in the Questa VM menu.
   This opens the Enter TestPlan Details dialog.
2. Enter the name of the testplan and the colors to use for the columns and select OK.

   This opes a file browser to choose the UCDB file to use for the generation.
3. Find the UCDB file you wish to use, and select **OK** to generate the testplan spreadsheet. All functions of the original spreadsheet are now available to use with the new testplan spreadsheet.

Using an Existing Excel Testplan as a Template

As long as an existing spreadsheet follows a similar format to the format supported within Questa, the Add-In functions should work without having to re-format the spreadsheet. This means that any testplan spreadsheet saved in .XML, .XLS, or .XLSX should work with the Add-In, providing that the format of the testplan is consistent with Questa's Excel template. The Add-In scans the worksheet to ensure that there is a title row containing: In the first column of the spreadsheet, a section number (labeled "Section" or "#") and in the second column, a section title (labeled "Section" or "Title"); if these are not found, the functions will not run.

If the spreadsheet you want to use doesn't follow the correct/expected format, the best course of action is to:

1. Import spreadsheet into a UCDB file in the way you have been using within your Questa flow.
2. Follow the instructions in "Generating a Testplan from a Testplan UCDB" to use the testplan UCDB file as the input for the generation of a new testplan spreadsheet.

Managing Links

When you use the Add Link data entry dialog within the toolbar — to help with the linking to the coverage objects — an association is made with a UCDB file automatically. Once this association has been made, the Add-In generates cache link files from the UCDB and stores them in a tmp directory named:

<Windows Local Drive>:\Users\Public\QuestaExcelAddInCache

These files are updated by the Add-In when the UCDB file changes.

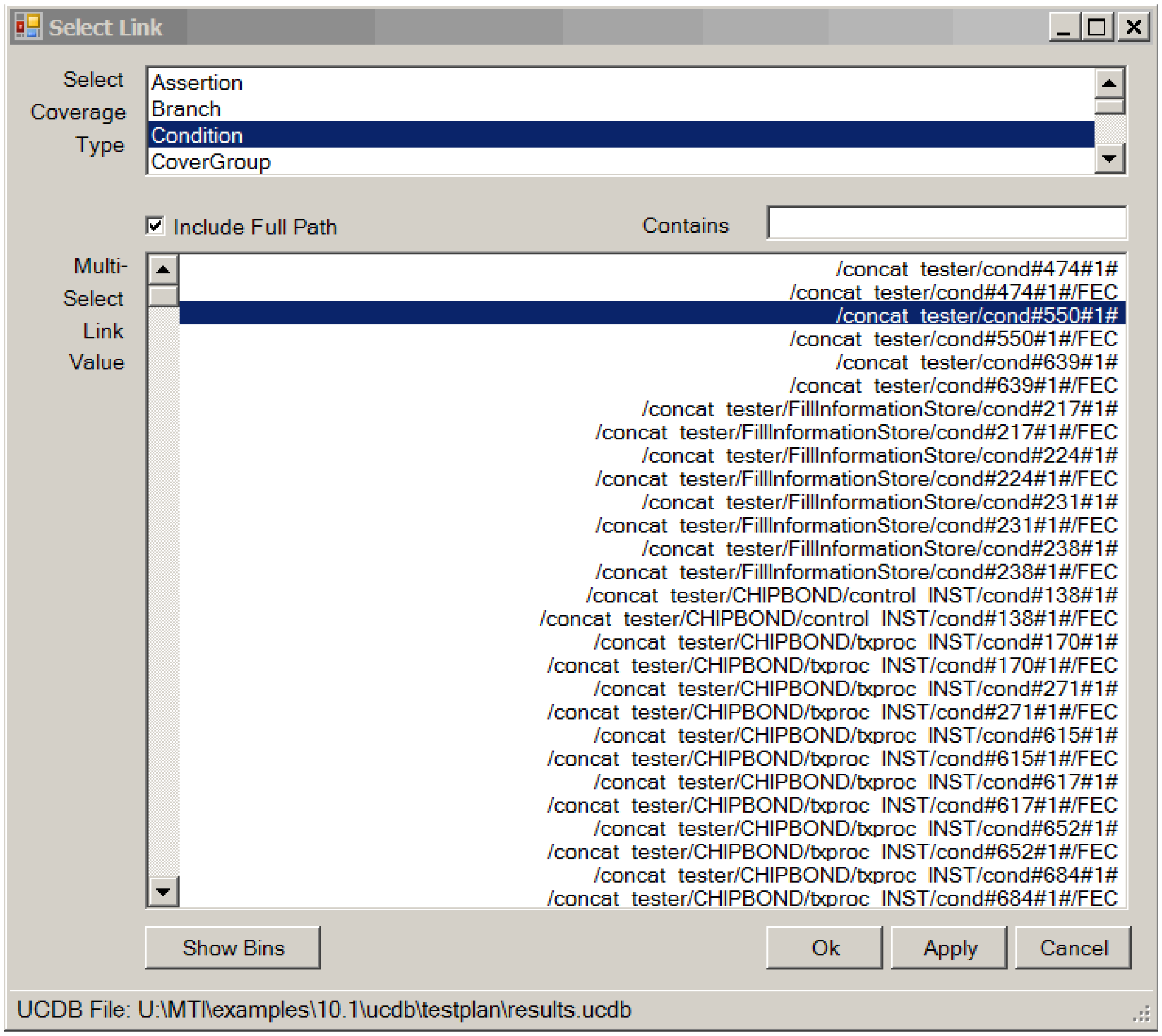
You can also control the type of coverage that is read in to the spreadsheet from the UCDB file. In some circumstances with very large coverage data sets, it may be better to reduce the amount of data loaded. You can control the import of functional, assertion and code coverage separately through the "Settings Dialog".

Adding Links with the Select Link Dialog

You can add links to the selected cell in a spreadsheet using the Select Links dialog, as follows:

1. Select a cell in of the spreadsheet in the Link column that needs to be filled.
2. Select the "Add Link" button. If a UCDB file has not been selected, a file selection dialog opens to allow you to choose the UCDB file which has the coverage link information that you require. If a UCDB file is selected, the Select Link dialog box appears.

Figure 3-11. Select Links Dialog



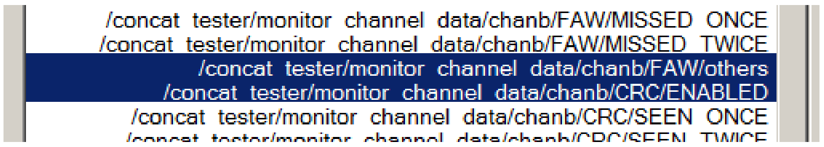
```
/concat_tester/cond#474#1#
/concat_tester/cond#474#1#/FEC
/concat_tester/cond#550#1#
/concat_tester/cond#550#1#/FEC
/concat_tester/cond#639#1#
/concat_tester/cond#639#1#/FEC
/concat_tester/FillInformationStore/cond#217#1#
/concat_tester/FillInformationStore/cond#217#1#/FEC
/concat_tester/FillInformationStore/cond#224#1#
/concat_tester/FillInformationStore/cond#224#1#/FEC
/concat_tester/FillInformationStore/cond#231#1#
/concat_tester/FillInformationStore/cond#231#1#/FEC
/concat_tester/FillInformationStore/cond#238#1#
/concat_tester/FillInformationStore/cond#238#1#/FEC
/concat_tester/CHIPBOND/control_INST/cond#138#1#
/concat_tester/CHIPBOND/control_INST/cond#138#1#/FEC
/concat_tester/CHIPBOND/txproc_INST/cond#170#1#
/concat_tester/CHIPBOND/txproc_INST/cond#170#1#/FEC
/concat_tester/CHIPBOND/txproc_INST/cond#271#1#
/concat_tester/CHIPBOND/txproc_INST/cond#271#1#/FEC
/concat_tester/CHIPBOND/txproc_INST/cond#615#1#
/concat_tester/CHIPBOND/txproc_INST/cond#615#1#/FEC
/concat_tester/CHIPBOND/txproc_INST/cond#617#1#
/concat_tester/CHIPBOND/txproc_INST/cond#617#1#/FEC
/concat_tester/CHIPBOND/txproc_INST/cond#652#1#
/concat_tester/CHIPBOND/txproc_INST/cond#652#1#/FEC
/concat_tester/CHIPBOND/txproc_INST/cond#684#1#
/concat_tester/CHIPBOND/txproc_INST/cond#684#1#/FEC
```

UCDB File: U:\MTI\examples\10.1\ucdb\testplan\results.ucdb

3. Select the coverage type for the link you want to add from the Select Coverage Type list.
4. Select one or more links from the Multi-Select Link Value list.
5. To see and/or link bins for the coverage type selected:
    a. If the selected Coverage Type is anything other than **Du**, check the Include Full Path checkbox.
    b. Select the Show Bins button.

       The Select Bin Link dialog opens, containing a list of all bins associated with the link value(s) selected.
    c. To link a bin or bins to the spreadsheet row, select one or more bins from the list. The showbins button is greyed out if the Coverage Type is **Test**.

Figure 3-12. Selecting Bins for Linking



```
/concat tester/monitor channel data/chanb/FAW/MISSED ONCE
/concat tester/monitor channel data/chanb/FAW/MISSED TWICE
/concat tester/monitor channel data/chanb/FAW/others
/concat tester/monitor channel data/chanb/CRC/ENABLED
/concat tester/monitor channel data/chanb/CRC/SEEN ONCE
```

d. Select either **Apply** or **OK** in the Select Bin Link dialog.

   OK adds the link(s) and dismisses the dialog, where Apply adds the link(s) and leaves the dialog open for more additions.

6. Select either the **Apply** or **Ok** button in the dialog to add the links and types into the cells of the selected row. **Ok** also dismisses the dialog.

   As many selections of different coverage types that are made, the links and types are added to the cells on each press of the **Apply** button.

   For each addition of a link, the entry within the cell in the Type column is adjusted to ensure that the rules which xml2ucdb require for multiple coverage types are preserved.

Removing Existing Association with Testplan

If a currently active spreadsheet is already associated to a particular testplan other than the one you want to create links to, you must first remove that association (link) before you can select another UCDB for linking.

To remove the association, select:

VM menu > **Manage UCDB Links** > **Remove**

Saving and Exporting to XML and UCDB

You can save and export in both XML and UCDB formats. The detailed instructions of each, as well the advantages, are contained in this section.

How and Why to Export TestPlan to XML

The **Save/Export Testplan** > **to XML** menu selection exports an XML file that can be used to import the testplan into Questa via the xml2ucdb program. The XML is output in a format that can be processed using the Excel format default in the xml2ucdb.ini of the Questa SIM product installation.

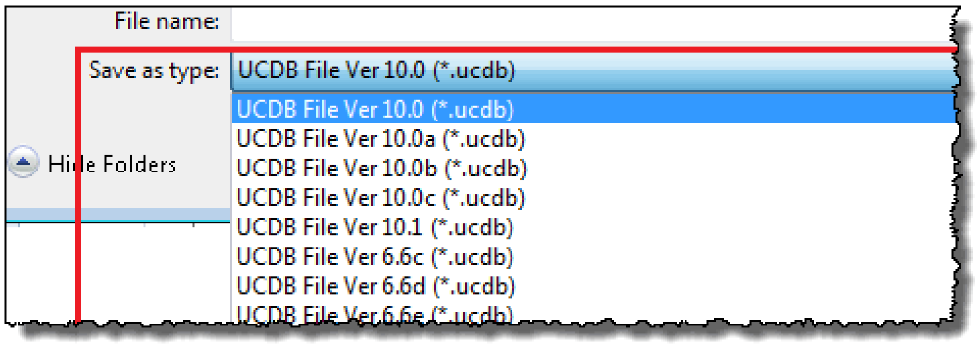
If your testplan contains any columns that are not part of the default setting of the *xml2ucdb.ini* file, you must edit the -datafields parameter to match the additions and order that exist within the plan spreadsheet. See the chapter entitled “XML2UCDB.ini File” in the “Questa Verification Management User’s Manual” for details on setting the -datafields parameter within the xml2ucdb.ini file, and the “Questa Reference Manual” for xml2ucdb syntax.

The xml2ucdb function transverses the testplan data within the spreadsheet and generates an XML file. The benefit of this method is that the testplan spreadsheet itself doesn't need to be saved as XML, it can be kept as an Excel binary file, and the output from this process can be used to import the data to UCDB (through xml2ucdb).

How and Why to Export Testplan to UCDB

The **Save/Export Testplan > to UCDB** menu selection not only exports an XML file, but it also generates a script to run the xml2ucdb executable and then runs the xml2ucdb executable to produce a UCDB with the testplan imported. This method of exporting the spreadsheet has the advantage of automatically interpreting and adding to the UCDB any additional columns found in the spreadsheet above and beyond the default columns that the Questa SIM xml2ucdb tool expects.

- It is very important to correctly set the version of the UCDB file. Questa is backward compatible. This means that if you generate a 10.1 UCDB and try to open it with Questa 10.0, it will not work. Take care to ensure you sure to select the correct version of the UCDB for Excel to use in the **Save as type:** pull-down menu, within the **Save As** dialog box.


File name:
Save as type:
UCDB File Ver 10.0 (*.ucdb)
UCDB File Ver 10.0 (*.ucdb)
UCDB File Ver 10.0a (*.ucdb)
UCDB File Ver 10.0b (*.ucdb)
UCDB File Ver 10.0c (*.ucdb)
UCDB File Ver 10.1 (*.ucdb)
UCDB File Ver 6.6c (*.ucdb)
UCDB File Ver 6.6d (*.ucdb)
UCDB File Ver 6.6e (*.ucdb)
Hide Folders

- Skipping Columns in Plan — Any columns not formatted in bold are essentially skipped in the created testplan. The UCDB column generation function scans the title row of the spreadsheet and builds the datafield entry from your spreadsheet, testing all titles and only including them for import when its format type is "bold". For all non-bold titles, it adds a "-" into the datafield entry for that column, such that the column is skipped by the import process.

Note

The import command generated in the script to run the process does not use the -format switch. This is done so that it doesn't need to read an xml2ucdb.ini file, as then it would require the datafield entry to be modified. All the settings for the xml2ucdb process are done with xml2ucdb command line switches and is the reason behind the import log having a warning that no -format switch was used. This can be ignored.

- To create unimplemented links, or nest XML testplans, see the "Settings Dialog".

Validation Checks Performed

A number of syntax or format problems can occur during the development of a testplan in the spreadsheet. These problems are normally found during the use of the xml2ucdb process, with errors and warnings printed as part of the process. However, you can select the “Validate Testplan” menu item to check the format and syntax within your spreadsheet.

The checks carried out by the validation process are as follows;

1. Section numbers repeated — ensures that no two testplan sections contain the same number. This would cause overwrites within the UCDB.
2. Section numbers in the wrong order — ensures that the testplan sections are in an increasing count. This helps to make sure that parent sections assist first.
3. Section names repeated — ensures that the same name is not used for a section name. The section name is used as the identifier within the UCDB so two children sections cannot have the same name.
4. Empty weight columns— ensures that the weight section has an entry. Blank weights import with a weight of 1, which means that coverage items corresponding to that section of the plan are included in testplan coverage calculations.
5. Empty goal columns — ensures that the goal section has an entry. Blank goals import as a 100 percentage goal.
6. Unmatched Link and type fields — ensures that there is either the same number of links as types or a single type associated with all links. Unmatched links and types cause an error when the xml2ucdb command is run.
7. Invalid Type entries — ensures that all the link types in the Type column are valid. Valid Types include: Assertion, Branch, Condition, CoverGroup, CoverPoint, Cross, Directive, Du, Expression, FSM, Instance, Test, Toggle, XML, Tag, and CoverItem. See “Linkable Design Constructs” in the Questa SIM Verification Management User’s Manual for details.

A message dialog pops up when the validation process is complete.

Chapter 4
Automated Run Management

Introduction to Run Management

In a coverage driven verification methodology, when a new IP for a SoC or FPGA design is being developed, the verification team starts the verification process by creating a plan that describes:

- What will be verified - Prioritized list of key features
- How it will be verified - Simulation, Formal Emulation, FPGA Prototyping and for simulation, techniques such as constrained Random, Directed test techniques, etc.
- How progress will be measured - Usually in the form of functional coverage, assertion coverage, code coverage, coverage from Formal engines, etc.

The team creates the executable platform, including coverage models with which to exercise the design under verification, in order to answer the questions posed in the verification plan. Then, they proceed to verify the DUV by executing this platform.

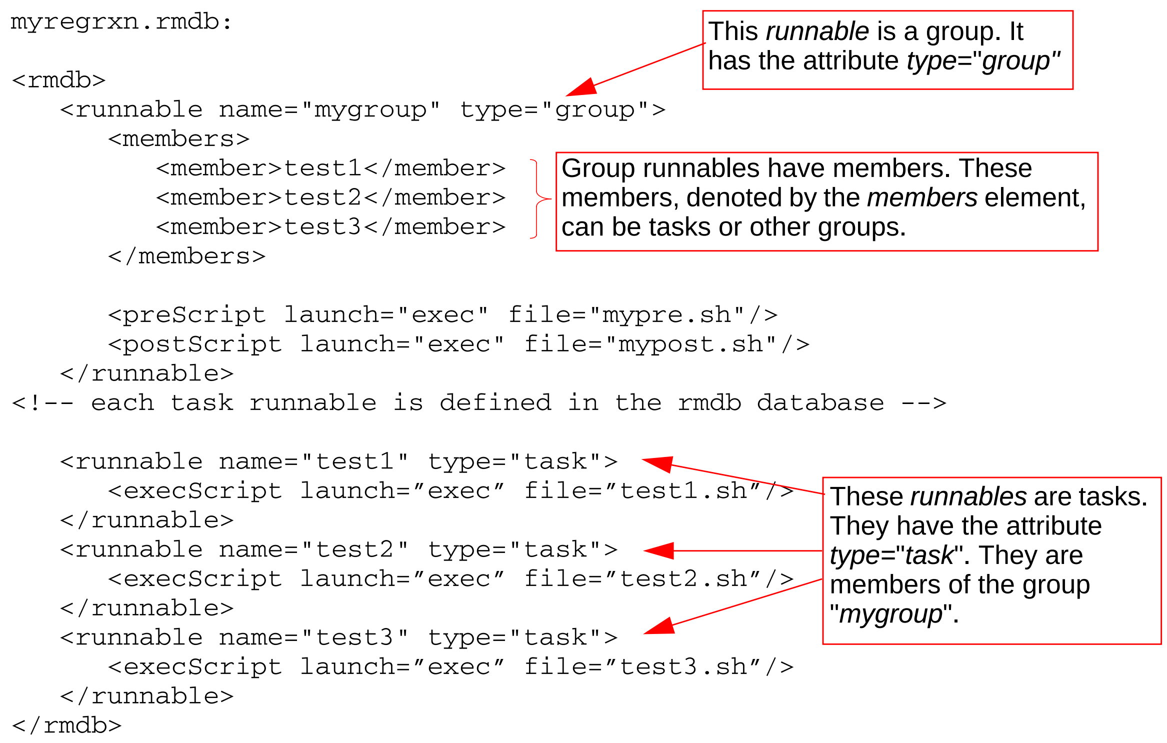
Typically, the verification process involves a large number of regression cycles and could involve multiple tools, each with its own generated metrics. It is not unusual to see verification teams getting swamped with having to maintain such a process, rather than focusing on the their primary function of verification.

The Verification Run Manager VRM accelerates the verification process through use of automation and an understanding of the typical verification process. It allows the user to easily describe their regression suite. This description is called the Run Manager Database RMDB. The user can create highly parameterizable and portable regression environments which can be reused in many different ways. In addition, VRM provides hooks such that the user can better adapt it in their environment.

Run Management Examples

Example 4-1 is a kind of a template/example of a RMDB file.

Example 4-1. RMDB File Example



```
myregrxn.rmdb:

<rmdb>
   <runnable name="mygroup" type="group">
      <members>
         <member>test1</member>
         <member>test2</member>
         <member>test3</member>
      </members>

      <preScript launch="exec" file="mypre.sh"/>
      <postScript launch="exec" file="mypost.sh"/>
   </runnable>
<!-- each task runnable is defined in the rmdb database -->

   <runnable name="test1" type="task">
      <execScript launch="exec" file="test1.sh"/>
   </runnable>
   <runnable name="test2" type="task">
      <execScript launch="exec" file="test2.sh"/>
   </runnable>
   <runnable name="test3" type="task">
      <execScript launch="exec" file="test3.sh"/>
   </runnable>
</rmdb>
```

This *runnable* is a group. It has the attribute *type*="*group*"

Group runnables have members. These members, denoted by the *members* element, can be tasks or other groups.

These *runnables* are tasks. They have the attribute *type*="*task*". They are members of the group "*mygroup*".

Following is the command to launch this example:

vrun -rmdb myregrxn.rmdb mygroup

Complete Example

To view and run a complete yet simple example of the VRM, see *<install_dir>/vrm/fpu_ovm*. The RMDB for this example is shown in Example 4-2.

Detailed information on the use of the VRM can be found in the *Verification Run Management Manual*.

Example 4-2. fpu_conf.rmdb

```
<?xml version="1.0"?>
<rmdb loadtcl="mytcl" >

  <!-- ================================================ -->
  <!-- == Top Level Group Runnable  =================== -->
  <!-- ================================================ -->
  <runnable name="flow" type="group" sequential="yes" >
    <parameters>
      <!-- <parameter name="NUM_SIM" ask="Enter number of simulation repeats :
   " accept="num(1,500)">2</parameter>  -->
      <parameter name="NUM_SIM">2</parameter>
      <parameter name="TESTCASE">fpu_test_sequence_fair fpu_test_neg_sqr_sequence
          fpu_test_random_sequence fpu_test_simple_sanity</parameter>
      <parameter name="OVM_VERBOSITY_LEVEL">OVM_FULL</parameter>
      <parameter name="MODELSIM_INI"        >(%DATADIR%)/modelsim.ini</parameter>
      <parameter name="DPI_HEADER_FILE"     >(%DATADIR%)/dpiheader.h</parameter>
      <parameter name="SHARED_OBJECT_NAME"  >fpu_tlm_dpi_c</parameter>
      <parameter name="SHARED_OBJECT_FILE"
>(%DATADIR%)/(%SHARED_OBJECT_NAME%).so</parameter>
      <!-- If ucdbfile parameter is defined and mergefile parameter is
          defined, VRM will perform automatic merging of ucdb filed defined
          in ucdbfile into mergefile for passing tests. The user can specify
          mergeoptions optional parameter and mergescript action script but
          this is usually not necessary -->
      <parameter name="mergefile"           >fpu_trackr.ucdb</parameter>
      <!-- If ucdbfile parameter is defined and triagefile parameter is
          defined, VRM will perform automatic triage database creation of
          failing tests. It uses ucdbfile to locate the WLF file that it
          uses. The user can specify triageoptions optional parameter which
          can be useful if message transformation is desired -->
      <parameter name="triagefile"          >fpu_triage.tdb</parameter>
      <parameter name="triageoptions"       >-severityAll -teststatusAll -r
(%CONFDIR%)/transformrule.txt -deletemsg {{IGNORE}}</parameter>
      <!-- If ucdbfile, mergefile and tplanfile parameters are defined VRM
          will perform automatic testplan import and merging into mergefile
          as the first action of running a regression. The user can specify
          optional tplanoptions parameter for control of the testplan import
          process -->
      <parameter name="tplanfile"
>../fpu/vplan/vplan_excel.xml</parameter>
      <parameter name="tplanoptions"
>-format Excel -startstoring 3</parameter>

      <parameter name="QUESTA_HOME">(%MODEL_TECH%)/..</parameter>
      <parameter name="QUESTA_OS" >linux</parameter>

      <parameter name="OVM_VERSION">ovm-2.0.3</parameter>

      <parameter name="VLIB"      type="tcl">[file join (%QUESTA_HOME%)
       (%QUESTA_OS%) vlib]</parameter>
      <parameter name="VMAP"      type="tcl">[file join (%QUESTA_HOME%)
       (%QUESTA_OS%) vmap]</parameter>
      <parameter name="VCOM"      type="tcl">[file join (%QUESTA_HOME%)
       (%QUESTA_OS%) vcom]</parameter>
      <parameter name="VLOG"      type="tcl">[file join (%QUESTA_HOME%)
       (%QUESTA_OS%) vlog]</parameter>
      <parameter name="VOPT"      type="tcl">[file join (%QUESTA_HOME%)
       (%QUESTA_OS%) vopt]</parameter>
      <parameter name="VSIM"      type="tcl">[file join (%QUESTA_HOME%)
       (%QUESTA_OS%) vsim]</parameter>
      <parameter name="XML2UCDB"  type="tcl">[file join (%QUESTA_HOME%)
       (%QUESTA_OS%) xml2ucdb]</parameter>
      <parameter name="VCOVER"    type="tcl">[file join (%QUESTA_HOME%)
       (%QUESTA_OS%) vcover]</parameter>

      <parameter name="DUT_LIB"   type="file">(%DATADIR%)/dut_lib</parameter>
      <parameter name="WORK_LIB"  type="tcl">[file join (%DATADIR%)
   work]</parameter>
      <parameter name="OVM_LIB"   type="tcl">[file join (%QUESTA_HOME%)
       (%OVM_VERSION%)]</parameter>
    </parameters>
    <members>
      <member>compile</member>
      <member>sim_group</member>
      <member>vm</member>
    </members>
    <method name="grid" gridtype="sge" mintimeout="600" if="{(%MODE:%)} eq
       {grid}">
      <command>qsub -V -cwd -b yes -e /dev/null -o /dev/null -N (%INSTANCE%)
       (%WRAPPER%)</command>
    </method>
  </runnable>
```

```
<!-- ============================================== -->
<!-- == Compile group Runnable                   == -->
<!-- == This group is sequential. A failure in any == -->
<!-- == member of this group will cause the entire == -->
<!-- == group and downstream operations to be      == -->
<!-- == abandoned. Runnables are being used here   == -->
<!-- == for more control over specification of     == -->
<!-- == dependencies                               == -->
<!-- ============================================== -->
<runnable name="compile" type="group" sequential="yes">
  <parameters>
    <parameter name="DUT_SRC"  export="yes" type="file">../fpu</parameter>
    <parameter name="TB_SRC"   export="yes" type="file">../tb</parameter>
    <parameter name="OVM_HOME" export="yes" type="tcl">[file join
        (%QUESTA_HOME%) verilog_src (%OVM_VERSION%)]</parameter>
  </parameters>
  <members>
    <member>compile_init</member>
    <member>compile_elements</member>
    <member>optimize</member>
  </members>
</runnable>

<!-- create libraries, create modelsim.ini file and
     copy modelsim.ini file to a known location -->
<runnable name="compile_init" type="task">
  <execScript launch="/bin/csh -f">
    <command>(%VLIB%) (%WORK_LIB%)</command>
    <command>(%VLIB%) (%DUT_LIB%)</command>
    <command>(%VMAP%) -c mtiOvm (%OVM_LIB%)</command>
    <!-- The modelsim.ini file generated from vmap is
         copied to a location specified by a parameter.
         A link from this location to the task directories
         that need it is made use a localfile element                     -->
    <command>cp modelsim.ini (%MODELSIM_INI%) </command>
    <!-- Need to unset the MODELSIM environment variable to prevent
         background compilation from picking a different modelsim.ini
         file than the local version. This issue seems to occur when
         VRM and vsim versions are different.                            -->
    <command>unsetenv MODELSIM </command>
  </execScript>
</runnable>

<!-- DUT and TB do not have dependencies so can be launched in parallel  -->
<runnable name="compile_elements" type="group" sequential="no">
  <members>
    <member>compile_dut</member>
    <member>compile_tb</member>
  </members>
</runnable>

<!-- DUT is compiled but first a symbolic link to %MODELSIM_INI% is made
     by VRM in this specific task based on localfile element content       -->
<runnable name="compile_dut" type="task">
  <parameters>
    <parameter name="VCOM_ARGS">-novopt -cover sbectf</parameter>
    <parameter name="VOPT_ARGS">-bbox</parameter>
    <parameter name="FILELIST" type="file">vhdl.f</parameter>
  </parameters>
  <localfile type="link" src="(%MODELSIM_INI%)" />
  <execScript launch="/bin/csh -f">
    <command>(%VCOM%) -work (%DUT_LIB%) (%VCOM_ARGS%) -f
 (%FILELIST%)</command>
    <command>(%VOPT%) -work (%DUT_LIB%) (%VOPT_ARGS%) fpu -o bbox </command>
  </execScript>
</runnable>

<!-- TB System compilation group including compilation of DPI using gcc -->
<runnable name="compile_tb" type="group" sequential="yes">
```

```
  <parameters>
    <parameter name="DO_COMPILE_OVM">0</parameter>
  </parameters>
  <members>
    <member>compile_ovm_pkg</member>
    <member>compile_ovm_tb</member>
    <member>compile_reference_model</member>
  </members>
</runnable>

<!-- This is an example of a conditional runnable.
     Compile the open kit OVM into the work library if parameter
     DO_COMPILE_OVM is set to 1 otherwise use default precompiled
     closed kit OVM in Questa                                  -->
<runnable name="compile_ovm_pkg" type="task" if="(%DO_COMPILE_OVM%)==1">
  <parameters>
    <parameter name="FILELIST" type="file">(%OVM_VERSION%).f</parameter>
  </parameters>
  <localfile type="link" src="(%MODELSIM_INI%)" />
  <execScript launch="/bin/csh -f">
    <command>(%VLOG%) -work (%WORK_LIB%) (%VLOG_ARGS%) -f (%FILELIST%)</command>
  </execScript>
</runnable>


<runnable name="compile_ovm_tb" type="task">
  <parameters>
    <parameter name="VLOG_ARGS">-novopt -debugCellOpt -suppress 2181 -
 Epretty vlog.sv -dpiheader (%DPI_HEADER_FILE%)</parameter>
    <parameter name="PLUS_ARGS">+define+SVA+SVA_DUT</parameter>
    <parameter
 name="PLUS_ARGSbad">+define+SVA+SVA_DUT+FPU_TB_BUG</parameter>
    <parameter name="FILELIST" type="file">vlog.f</parameter>
  </parameters>
  <localfile type="link" src="(%MODELSIM_INI%)" />
  <execScript launch="/bin/csh -f">
    <command>(%VLOG%) -work (%WORK_LIB%) (%VLOG_ARGS%)
 (%PLUS_ARGS(%SETUP:%)%) -f (%FILELIST%)  </command>
  </execScript>
</runnable>

<runnable name="compile_reference_model" type="task">
  <parameters>
    <parameter name="REF_SRC"  type="file">.. fpu tlm c</parameter>
  </parameters>
  <localfile type="link" src="(%DPI_HEADER_FILE%)" />
  <execScript launch="/bin/csh -f">
    <command> g++ -g -c -m32 -fPIC -Wall -ansi -pedantic -I. -
 I(%QUESTA_HOME%)/include (%REF_SRC%)/(%SHARED_OBJECT_NAME%).cpp -o
 (%SHARED_OBJECT_NAME%).o </command>
    <command> g++ -shared -lm -m32 -Wl,-Bsymbolic -Wl,-export-dynamic -o
 (%SHARED_OBJECT_NAME%).so (%SHARED_OBJECT_NAME%).o </command>
    <!-- ensure that the shared obj is ready for the simulation runnable -->
    <!-- A localfile link can be used to refer to this later             -->
    <command> cp (%SHARED_OBJECT_NAME%).so (%SHARED_OBJECT_FILE%)</command>
  </execScript>
</runnable>

<runnable name="optimize" type="task">
  <parameters>
    <parameter name="VOPT_ARGS">+acc</parameter>
  </parameters>
  <localfile type="link" src="(%MODELSIM_INI%)" />
  <execScript launch="/bin/csh -f">
    <command>(%VOPT%) -work (%WORK_LIB%) -L (%DUT_LIB%) (%VOPT_ARGS%) top -o
 optimized </command>
  </execScript>
</runnable>
```

```
    <!-- ================================================ -->
    <!-- == Sim_group group Runnable                   == -->
    <!-- == This group runnable is used to wrap the    == -->
    <!-- == tasks and groups that make up the          == -->
    <!-- == simulation. Since there are no             == -->
    <!-- == dependencies between simulation tasks they == -->
    <!-- == can be launched concurrently on the GRID   == -->
    <!-- == in order to maximize regression throughput == -->
    <!-- ================================================ -->
    <runnable name="sim_group" type="group">
        <members>
          <member>simulate</member>
        </members>
    </runnable>

    <!-- An example of a repeat runnable. Simulate group will be repeated
         %NUM_SIM% times with different random seeds in this example   -->
    <runnable name="simulate" type="group" repeat="(%NUM_SIM%)">
      <parameters>
        <!--The parameter named seed has special behavior in VRM
            When a re-run is requested by the user, VRM will automatically
            replace the seed value "random" with the actual seed
            used in the first simulation run                           -->
        <parameter name="seed">random</parameter>
      </parameters>
      <members>
        <member>simulation</member>
      </members>
    </runnable>

    <!-- An example of a foreach runnable task. The simulation task will Iterate
         over the a list of test cases. In this example, these testcases are
         different tests derived from ovm_test                          -->
    <runnable name="simulation"  type="task" foreach="(%TESTCASE%)">
      <parameters>
        <!-- The definition of parameter named ucdbfile will cause automatic
     pass / fail determination to be based on value of ucdb attribute
     TESTSTATUS in VRM in addition to its primary task of specifying
     the location of ucdbfile -->
        <parameter name="ucdbfile">(%ITERATION%).ucdb</parameter>
        <parameter name="UCDBFILE"    export="yes">(%ucdbfile%)</parameter>
        <parameter name="VSIM_DOFILE" type="file">scripts/vsim.do</parameter>
        <parameter name="VSIMSWITCHES"> -onfinish stop -suppress 8536 -wlf
     (%ITERATION%).wlf -assertdebug -coverage -displaymsgmode
     both</parameter>
        <parameter name="VSIMARGS">(%VSIMSWITCHES%) -do "source (%VSIM_DOFILE%)"
     -sv_lib (%SHARED_OBJECT_NAME%) -lib (%WORK_LIB%)</parameter>
      </parameters>
      <localfile type="copy" src="scripts/wave.do"/>
      <localfile type="copy" src="scripts/wave_batch.do"/>
      <localfile type="link" src="(%MODELSIM_INI%)"/>
      <localfile type="link" src="(%SHARED_OBJECT_FILE%)"/>
      <execScript launch="/bin/csh -f"  mintimeout="300">
        <command>(%VSIM%) -c (%VSIMARGS%) -sv_seed (%seed%)
  +OVM_TESTNAME=(%ITERATION%) +OVM_VERBOSITY_LEVEL=(%OVM_VERBOSITY_LEVEL%)
  optimized</command>
      </execScript>
    </runnable>

    <!-- ================================================ -->
    <!-- == Reporting Runables  ========================= -->
    <!-- ================================================ -->
    <runnable name="vm" type="group" sequential="yes">
      <members>
        <member>vm_report</member>
      </members>
      <!--VRM vrun supports a macro mode which allows the user
          to invoke vrun from within a runnable’s action script -->
      <postScript >
        <command>vrun -vrmdata (%DATADIR%) -status -full -html -htmldir
     (%DATADIR%)/vrun</command>
      </postScript>
    </runnable>

    <runnable name="vm_report" type="task">
      <execScript >
        <command>vcover report -totals (%mergefile%) -file
     (%DATADIR%)/summary_cov.rpt</command>
        <command>triage report -name (%triagefile%) -file
     (%DATADIR%)/summary_triage.rpt</command>
      </execScript>
    </runnable>

    <!-- ============================================== -->
    <!-- == Over-ride TCL built-in                  == -->
    <!-- == Advanced VRM capabilities               == -->
    <!-- == The underlying default behaviour of VRM    == -->
    <!-- == can be modified using the exposed TCL API  == -->
    <!-- == Usually these defined TCL methods do       == -->
    <!-- == by default. These methods can be           == -->
    <!-- == overriden within the usertcl element of    == -->
    <!-- == RMDB. In addition the user can define TCL  == -->
    <!-- == procedures that they wish to call in       == -->
    <!-- == action scripts here also.                  == -->
    <!-- ============================================== -->
    <usertcl name="mytcl">
     <!-- An example of overriding the VRM procedure StopRunning
          In this case, VRM will stop if 1000 errors are generated
          during execution of a regression                   -->
     proc StopRunning {userdata} {
       upvar $userdata data
       set result [expr {$data(fail) == 1000}]
       return $result
     }

     <!-- An example of overriding the VRM procedure OkToMerge
          This example changes the default behavior such that all
          ucdbfiles of both passes and failures are merged.
          By default only passing ucdbfiles are merged        -->
     <!--proc OkToMerge {userdata} {
       upvar $userdata data
       return 1 ; approve merge.
     }-->

     <!-- An example of overriding the VRM procedure OkToTriage
          This example changes the default behavior such that all
          ucdbfiles of both passes and failures are used to create
          the triage database.
          By default only failing ucdbfiles are used for this purpose
          This is not recommended for real projects          -->
     proc OkToTriage {userdata} {
       upvar $userdata data
       return 1 ; approve triage.
     }
    </usertcl>
  </rmdb>
```

Chapter 5
Analyzing Test Results Based on a Plan

This chapter assumes a basic knowledge and understanding of the UCDB and the test data stored there. It does not contain this information, except as it relates to a verification plan.

To find information on the following topics, please see the "Coverage and Verification Management in the UCDB" chapter in the Questa SIM SV/AFV User's Manual:

- coverage algorithms for coverage summary
- merging tests
- ranking test results
- generating HTML and textual coverage summary reports

The information in this chapter focuses on how to analyze the Verification Plan data along with the test data from tests performed. It will demonstrate how to use the Verification Management tracker within Questa SIM as well as the command line query language.

The UCDB format can hold coverage information, test information and also testplan information, this means that the UCDB file that is used to track testplans has to have all this information merged into a single UCDB file that is loaded into Coverage View (with vsim -viewcov) mode. This in-memory UCDB file is then viewed within the Tracker tab of the Verification Management pane within the user interface.

The specific functionality discussed in this chapter is:

Preparing the UCDB File for Tracking

In Chapter 2, details were provided on how to import testplans written in document formats such as Excel or Word into the UCDB format. Once the testplan is imported into a UCDB file, it is necessary to merge it with the data stored in UCDB files from the multiple simulation or tool runs. The process of the merge is reactive to the data that is stored within the UCDB files. To merge coverage data from simulation or tool runs along with testplan data, you could use a command similar to the following:

```
vcover merge tracking.ucdb testplan.ucdb regression/*.ucdb
```

This command merges the data from all the simulation runs stored in the directory *regression* along with the *testplan.ucdb* database, which was the output from the xml2ucdb process as explained in the last chapter. *tracking.ucdb* is the final merged output UCDB file. The merge performed (by default) is a test-associated merge type. This is important because when the testplan is analyzed and queries are made about the effectiveness of tests, the information about the association between tests and coverage data is kept. For more information on the different merge algorithm options and how they impact coverage analysis, see "About the Merge Algorithm" and "Limitations of Merge for Coverage Analysis".

For detailed instructions for merging the testplan with the test data, as well as linking between the two, see "Combining Verification Plan Data and Test Data".

Viewing Test Data in the Browser Window

This information can be found in the section "Viewing Test Data in the Browser Window" in the Questa SIM User's Manual.

Viewing Test Data in the Tracker Window

The coverage analyze command — valid only in Coverage View mode (vsim -viewcov) — is the command on which the Tracker window is based. See this command in the Questa SIM Reference Manual for further details on the specific functionality.

Prerequisites

To view test data in the Verification Tracker window, you must have already:

- Imported a testplan into UCDB format
  See "Importing an XML Verification Plan".
- Merged your design test data with the verification plan into a single UCDB.
  See "Merging Verification Plan with Test Data".
- Merged file must be viewed in Coverage View mode.
  See "Invoking Coverage View Mode".

Procedure

1. Open the Verification Browser window, if it is not open already:

   View > Verification Management > Browser

2. Double-click the merged file (UCDB) that contains verification (test) plan and test results. If you do not have a merged UCDB that contains a testplan, see the Questa AFV User's Manual for instructions on how to merge UCDBs.

This opens the UCDB in the Tracker in Coverage View mode. Alternatively, you could do this explicitly:

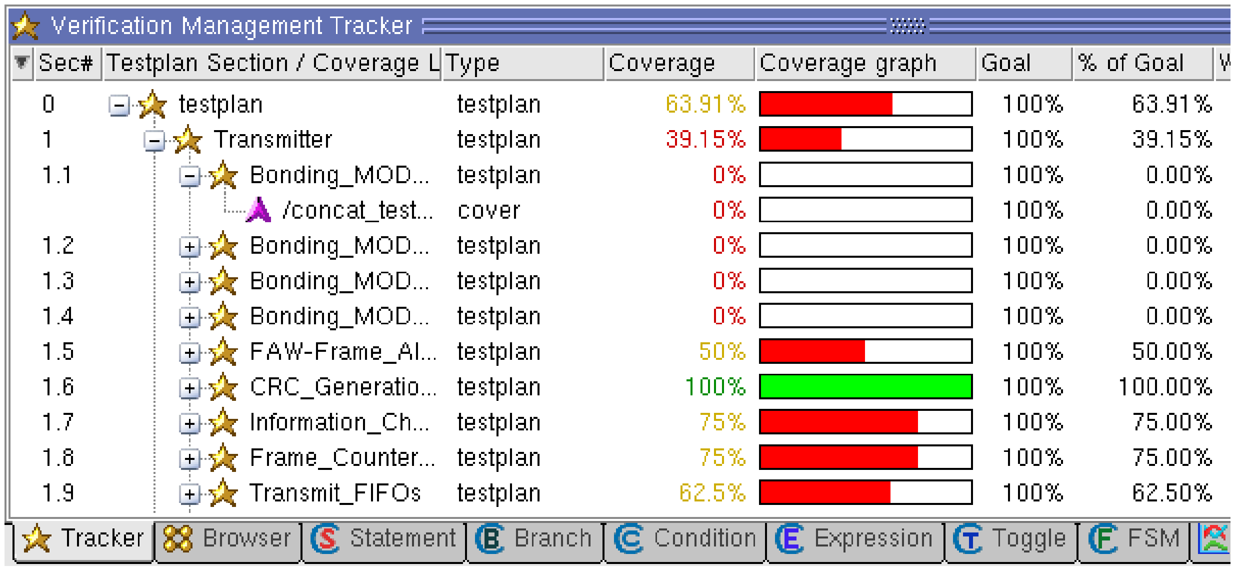
- from the command line, using:

  vsim -viewcov <merged.ucdb>

- in the GUI: right click anywhere in the Verification Browser, and select **Invoke CoverageView Mode**.

The Verification Tracker window appears, similar to Figure 5-1. The coverage statistics displayed in the Coverage column in the Tracker depend on whether the item is a design unit or an instance. See "Calculation for Total Coverage" for details.

Figure 5-1. Test Data in Verification Tracker Window



Sec#	Testplan Section / Coverage L	Type	Coverage	Coverage graph	Goal	% of Goal	W
0	testplan	testplan	63.91%	🟥	100%	63.91%	
1	Transmitter	testplan	39.15%	🟥	100%	39.15%	
1.1	Bonding_MOD...	testplan	0%	⬜	100%	0.00%	
	/concat_test...	cover	0%	⬜	100%	0.00%	
1.2	Bonding_MOD...	testplan	0%	⬜	100%	0.00%	
1.3	Bonding_MOD...	testplan	0%	⬜	100%	0.00%	
1.4	Bonding_MOD...	testplan	0%	⬜	100%	0.00%	
1.5	FAW-Frame_Al...	testplan	50%	🟥	100%	50.00%	
1.6	CRC_Generatio...	testplan	100%	🟩	100%	100.00%	
1.7	Information_Ch...	testplan	75%	🟥	100%	75.00%	
1.8	Frame_Counter...	testplan	75%	🟥	100%	75.00%	
1.9	Transmit_FIFOs	testplan	62.5%	🟥	100%	62.50%	

Related Topics

- coverage analyze command
- Understanding Stored Test Data in the UCDB
- Calculation for Total Coverage
- Importing an XML Verification Plan
- Invoking Coverage View Mode
- Refreshing Tracker after Changing a Plan
- Merging Verification Plan with Test Data
- Changing Goal, Weight and User Attribute Values in Tracker

Invoking Coverage View Mode

UCDB files from previously saved simulations are only viewable in Coverage View mode (post-processing). You can invoke Coverage View Mode on any of your .ucdb files in the Test Browser or at the command line. This allows you to view saved and/or merged coverage results from earlier simulations.

Procedure

- GUI:

  a. Right-click to select .ucdb file. This functionality does not work on .rank files.

  b. Select **Invoke CoverageView Mode**.

  The tool then opens the selected .ucdb file and reformats the Main window into the coverage layout. A new dataset is created.

- Command Line:

  Enter vsim with the -viewcov <ucdb_filename> argument. Multiple -viewcov arguments are allowed. For example, the Coverage View mode is invoked with:

  ```
  vsim -viewcov myresult.ucdb
  ```

  where *myresult.ucdb* is the coverage data saved in the UCDB format. The design hierarchy and coverage data is imported from a UCDB.

Related Topics

- Coverage View Mode and the UCDB
- Viewing Test Data in the Tracker Window

Customizing the Column Views

You can customize the display of columns in the Verification Browser or Tracker windows, and then save these views for later use.

Procedure

1. Select **[Create/Edit/Remove ColumnLayout...]** from the pull down list.

   This displays the Create/Edit/Remove Column Layout dialog box.

2. Enter a new name in the Layout Name text entry box.

3. OK

You can also add or modify pre-defined column arrangement from the Create/Edit/Remove Column Layout dialog box by adding columns to or removing them from the Visible Columns box as desired.

After applying your selections, the rearranged columns and custom layouts are saved and appear when you next open that column view in the Verification Browser or Tracker windows.

Tip: You can print the contents of the Tracker window, including filtered views, to an HTML file by selecting File > Export > HTML (see Viewing Verification Tracker Data in HTML).

Related Topics

- Test Attribute Records in the UCDB
- Changing Goal, Weight and User Attribute Values in Tracker

Changing Goal, Weight and User Attribute Values in Tracker

You can change the values for Goal, Weight and any user defined attributes in a .ucdb file by editing the values inside the Tracker window. This is helpful when iteratively running simulations, and tracking the accumulated coverage data in a testplan inside the Tracker window.

Procedure

1. In the Tracker window Double click on any Goal, Weight, or user-defined attribute value within the UCDB file. This highlights the cell allowing you to edit.
2. Type the new value into the cell.
3. Select <Enter>.
4. Optionally, save the changed values using **Tools > Coverage Save** or the coverage save command at the command line.

If you exit the session without saving, a dialog box pops up asking if you want to save the changes.

Note

Remember that once the changes to the UCDB plan have been saved, it will be out of step with the source plan.

Related Topics

- Importing an XML Verification Plan
- Storing User Attributes in the UCDB
- Filtering Results by User Attributes

Analyzing Results with coverage analyze

The coverage analyze command — valid only in Coverage View mode (vsim -viewcov) — is the command on which the Tracker window is based. This CLI command can be very useful in querying the complete UCDB. Queries can be based not only on the plan, but also on the design/testbench itself.

Some sample coverage analyze commands you can use to view and analyze the verification data are as follows:

- Open a UCDB for querying:

% vsim -c -viewcov tracker.ucdb

This opens the *tracker.ucdb* in Coverage View mode. The following confirmation appears in the transcript:

```
# tracker.ucdb opened as coverage dataset "tracker"
```

- View the coverage summary report for the verification plan at the top level:

  % coverage analyze -plan /

  A report is printed to the transcript such as:

```
#
# Total Coverage Report
# Sec#     Testplan Section / Coverage Link Coverage    Goal Weight
# ------------------------------------------------------------------
# 0        /testplan                      42.48%   100.00%     1
```

- View the test with the most coverage:

  % coverage analyze -plan / -coverage most

  A report is printed to the transcript listing the test name and the coverage:

```
# Tests with Most Coverage:
#
# /testplan
#     goodseedtest~-1776403150
#                               37.62%
```

- View the test with the most coverage within a design unit:

  % coverage analyze -path /concat_tester -coverage most

  A report is printed to the transcript listing the test with the most coverage within the specified design unit, along with the coverage number:

```
# Tests with Most Coverage:
#
# /testplan
#     goodseedtest~-1776403150
#                               58.25%
```

Refreshing Tracker after Changing a Plan

If you previously opened up a database in the Verification Tracker window, and you need to make a change to the associated verification plan, a shortcut is available to refresh the data shown in the Tracker:

1. Ensure that the changed plan is saved to XML, using the file name previously used when you imported the testplan.
2. Right-click anywhere in the Tracker window containing the imported testplan, and select **Reimport Testplan and Refresh**.

The Testplan Reimport and Refresh dialog box opens, populated with the necessary commands for performing a re-import and re-merge operation.

3. Select OK to refresh the contents of the imported testplan.

Storing User Attributes in UCDB

You can add your own attributes to a specified test record using the coverage attribute command, with a command such as:

```
coverage attribute -test testname -name Responsible -value Joe
```

This command adds the "Responsible" attribute to the list of attributes and values displayed when you create a coverage report on *testname.UCDB*. This shows up as a column when the UCDB is viewed in the Tracker pane.

For further information on attributes, see "Understanding Stored Test Data in the UCDB".

Filtering Results by User Attributes

One powerful feature for tracing verification requirements is the ability to filter your coverage results by attributes, added to the testplan for that purpose. For example, if your original testplan included such data fields as the engineer responsible for tests, or the priority level assigned a specific section of the plan, you can add these as attributes to the imported testplan and use them for filtering the coverage results. The command that corresponds to this functionality is the coverage analyze command.

Requirements

- User attributes must already exist in the original plan.
- Plan must be imported (see "Importing an XML Verification Plan").
- You must add the columns by which you wish to filter to the plan (see "Adding Columns to an Imported Plan").
- Plan must be merged with test results (see "Merging Verification Plan with Test Data").
- Verification Management Tracker pane must be open (see "Viewing Test Data in the Tracker Window").

Procedure

The following steps describe how to filter items for display on a specific column in the UCDB verification plan.

1. Select all tests to which you want to apply selection criteria.
2. Right-click in the Tracker pane and choose **Filter** > **Setup** from the popup menu.

This opens the Filter Setup dialog box.

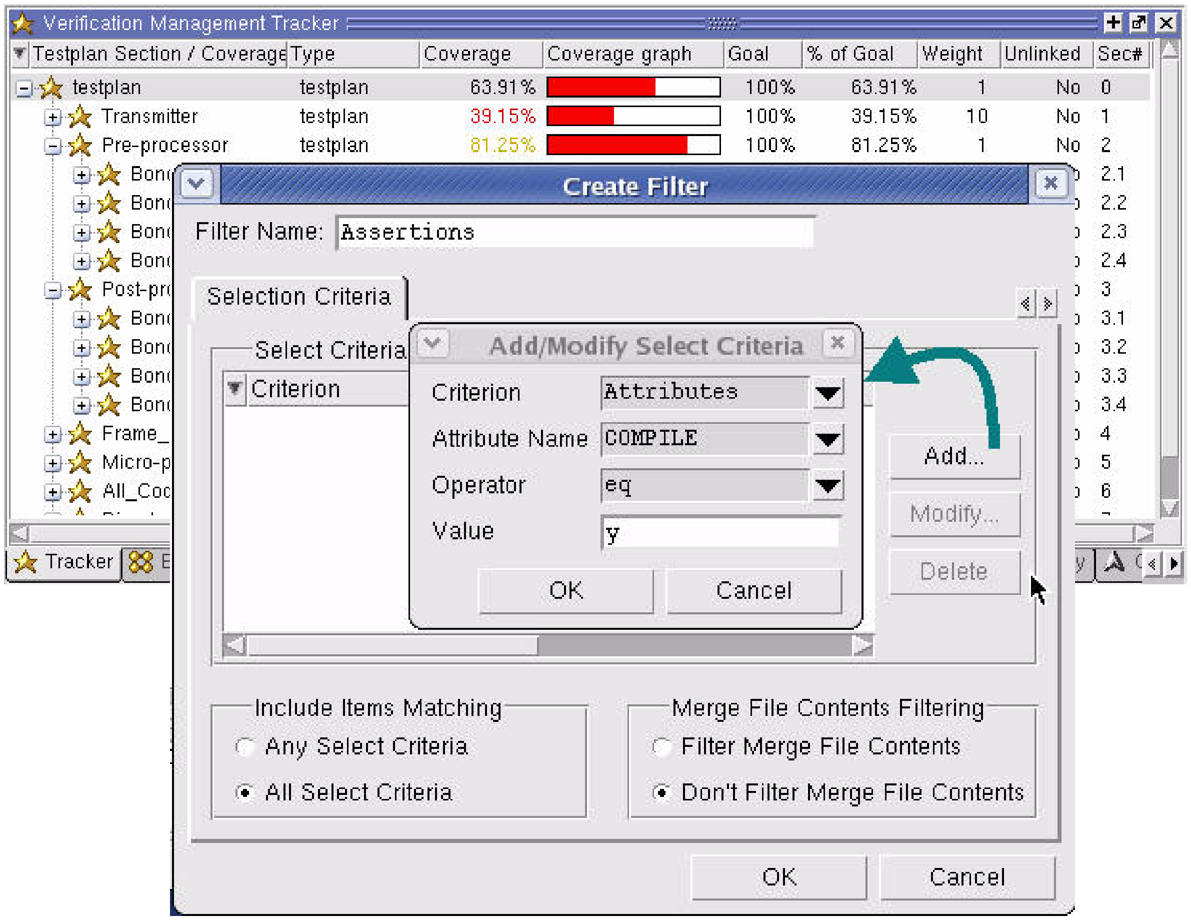
3. Click Create.

   This opens the Create Filter dialog box, shown in Figure 5-2.

4. Click Add.

   This opens the Add/Modify/Select Criteria dialog box.

Figure 5-2. Filtering Displayed Data



Testplan Section / Coverage	Type	Coverage	Coverage graph	Goal	% of Goal	Weight	Unlinked	Sec#
testplan	testplan	63.91%		100%	63.91%	1	No	0
Transmitter	testplan	39.15%		100%	39.15%	10	No	1
Pre-processor	testplan	81.25%		100%	81.25%	1	No	2

Filter Name: Assertions

Criterion: Attributes
Attribute Name: COMPILE
Operator: eq
Value: y

Include Items Matching: ( ) Any Select Criteria [✓ All Select Criteria]

Merge File Contents Filtering: ( ) Filter Merge File Contents [✓ Don't Filter Merge File Contents]

1. In the Criterion field, choose “attribute” from the pull-down list. Without selecting “attribute” from the pull-down list, the Attribute Name field will be grayed out.

   a. Select Attribute Name and select desired attribute name from pull-down list. The list contains all pre-defined attributes and any user attributes you added.

   b. Select Operator. See coverage analyze -select for definitions.

   c. Enter Value of item to match.

d. OK

The criterion you just entered appears in the Select Criteria list.

2. The Aggregate Coverage Totals section of the Create Filter dialog box offers you two options:

   - Before Making Selection (filtering) — coverage totals are aggregated before the selection criteria (filters) are applied, so all children's coverage numbers are considered in the coverage totals appearing in the testplan sections.

   - After Making Selection (filtering) — coverage totals are aggregated after the selection criteria (filters) are applied, so only surviving children's coverage numbers are considered in the coverage totals appearing in the testplan sections.

   If you select Before Making Selection, the Pruning section of the dialog box becomes active and the Prune Children or Include Children radio buttons can be selected. These control whether the filtering continues below the top level testplan sections:

   - Prune Children — recursively applies filtering to children of testplan sections that survived the filter; some children may be filtered out from display.

   - Include Children — leaves all children of testplan sections that survived the filter alone; all children remain included in display.

3. Select the **Specify Tests** tab to select specific tests. By default, all tests are subject to the filter.

4. Enter a **Filter Name** and select **OK** to save the filter with the specified selection criteria.

   The filter you just created appears in the Filters list within the Filter Setup dialog box.

5. Either select **Apply** to filter the UCDB data, or select Done to exit the dialog box.

Applying a Filter

- From the Filter Setup dialog, select the desired Filter from the list and select **Apply**.
- From the Verification Management Tracker:

  a. Right click on UCDB(s) to filter.

  b. Click on **Filter>Apply**, and then select a filter from the list.

  UCDBs with matching criteria are included in the data now displayed in the Browser.

Related Topics

- Understanding Stored Test Data in the UCDB
- coverage analyze Command
- xml2ucdb Command
- xml2ucdb.ini Configuration File
- Storing User Attributes in the UCDB

Retrieving Test Attribute Record Content

Two commands can be used to retrieve the content of test attributes: coverage attribute and vcover attribute, depending on which of the simulation modes you are in.

To retrieve test attribute record contents from:

- the simulation database during simulation (vsim), use coverage attribute. For example:

  ```
  coverage attribute
  ```

- a UCDB file during simulation, use vcover attribute. For example:

  ```
  vcover attribute <file>.ucdb
  ```

- a UCDB file loaded with -viewcov, use coverage attribute. For example:

  ```
  coverage attribute -test <testname>
  ```

The Verification Browser and Tracker windows display columns which correspond to the individual test data record contents, including name/value pairs created by the user. The pre-defined attributes that appear as columns are listed in Table 2-1.

See “Customizing the Column Views” for more information on customizing the column view.

Analysis for Late-stage ECO Changes

Often ECOs (Engineering Change Orders) can occur late in the design cycle, when a design is highly stable. Only small sections of the design are affected by changes. You can use various Verification Management tools to analyze which tests most effectively cover those few areas and re-run those specific tests to demonstrate satisfactory coverage numbers.

- Rank tests (see “Ranking Coverage Test Data”).
- Re-run tests (see “Rerunning Tests and Executing Commands”).

Chapter 6
Analyzing Verification Results

The Questa SIM Results Analysis functionality can be applied either within the Verification Run Manager (VRM) environment or in an independent scripting system. The "triage" set of CLI commands form the basis of Results Analysis, which brings together messages from multiple simulations and verification runs into a single view. This, in turn, allows you to group and store these messages to determine common failures and analyze these patterns. This capability of triaging data within a database serves the purposes of verification management, and is extremely useful in the analysis of the verification results.

This chapter includes:

Use Models for Results Analysis

Two predominant use models for test triage are regressions and checking individual simulations:

- Regressions —

  One main use of Results Analysis is to analyze the results of regressions run in a post-process step or within a script. After running regressions, a verification engineer typically scans through output files to look at the status (pass or fail) and messages. With sufficient checking, simulation passes are acknowledged while failures require deeper review to look for root causes. When the simulation set is large enough, the job of sifting through output files becomes tedious and time consuming. Results Analysis aims to streamline this root cause analysis by saving useful information in a database and providing viewing/reporting tools enabling a user to identify problems faster. Various filtering and hierarchical sorting tools allow the user to either find root causes of failures or choose priorities and resources for failure debug.

- Checking Individual Simulations —

  Open Results Analysis right after an individual simulation to automatically summarize failures. The desired triage commands can be added to the end of your simulation script or "do" file as a way to summarize any high-priority messages. In a situation where there is a test failure, the simulation may generate many messages, but the user may just see the last message. Sometimes, an earlier message is more useful in helping the user discover the failure's root cause. Instead of manually searching simulation output logs, the user can automate this process using Results Analysis.

Use Flow

The basic flow for Results Analysis would be:

1. Run one or more simulations/regressions to capture results, or otherwise obtain other plain-text file (transcript, 0-in, or 3rd party ASCII file).

   During simulation, Questa SIM writes messages into either a transcript log file (ASCII text) and/or a WLF file, depending on the method chosen.

   a. Merge any test results (UCDBs) you have into a single file for database creation with “vcover merge”.

2. The “triage dbfile” and “triage dbinsert” commands takes message data that has been saved to a WLF file or to a plain-text LOG file, and combines those messages with the UCDB attributes from the same simulation into the TDB (triage database) used for Results Analysis.

   a. As part of this step, you may need and/or want to create a Transform Rulesfile, containing one or more transform statements, to improve the usability of the data that is generated in the TDB.

      If you use a transcript or other plain-text file, you MUST create one or more transforms for message extraction. See “Transforms”.

3. Open Results Analysis window to query results in a variety of ways, sorting by severity and other criteria. See “Viewing Results Analysis Data”.

Input Files for Results Analysis Database

Three types of files are accepted as inputs to the Results Analysis database (TDB):

- UCDB File

  The UCDB file contains test data that is displayed and contains automatic mechanisms to extract associated messages:

  - o If the UCDB contains a WLFNAME attribute and the associated WLF has messages, it uses the filename to input messages. (Though a transform rule is not required in this case, it can be useful. See "Transformation of Messages" and "Transform Rulesfile" for more information.)
  - o If the UCDB does not contain a WLFNAME, the command attempts to automatically import a logfile through the use of a LOGNAME attribute. It uses the value of that attribute as a filename from which to inputs messages from Questa SIM. In this case, a transform is required.

  When you use a UCDB file as the input to Results Analysis, it also extracts the testname and displays that information. If you are not using a UCDB file, you must use the -testname argument to the "triage dbfile" command in order to properly extract the testname for the data.

- WLF file

  Since the time and messages have already been extracted into clear fields (i.e. "msg", "time" and "severity"), it is easy for the tool to group and/or sort this verification data. The "msg" field is the most important element of data, because its string is used for the first level grouping within triage. Transformation, in the case of WLF files, is useful though not required. See "Transformation of Messages".

- a transcript or plain text LOG file

  A plain-text log file — regardless of whether or not it was produced by Questa SIM, as in the case of a transcript, 0-in, VCS, or any 3rd party tool — must be processed using a rulesfile containing the necessary *transform* statement(s) in order to extract time, severity, and other types of information. See "Transforms" for more information.

Creating a Results Analysis Database (TDB)

Results Analysis helps to automate the process of reviewing test failures.

Prerequisites

- You must have captured test results (either in transcript log files (ASCII text) and/or WLF files, or UCDB files).
- Before creating the TDB, you may need and/or want to create a Transform Rulesfile, containing one or more transform statements, to improve the usability of the data that is generated in the TDB.

  If you use a transcript or other plain-text file, you MUST create one or more transforms for message extraction. See "Input Files for Results Analysis Database" for details on each of these file types, and to determine if transforms are required for the input files you are using.

Procedure

1. Create a triage database in the GUI or at the command line:
    - Command Line:

      ```
      triage dbfile <input>
      ```

      where <input> is one of the following:

        - a UCDB file
        - a WLF file
        - a plain-text LOG file
    - GUI:
        i. Open the GUI
        ii. From the menu bar, select **View** > **Verification Management** > **Results Analysis**
        iii. From the menu bar, select **Results Analysis** > **Create Database**

      This opens the Create Triage Database dialog box with fields matching the switches and arguments described in the triage dbfile command.

Related Topics

- Input Files for Results Analysis Database
- Triage Command Examples
- Transformation of Messages
- Verification Results Analysis Window
- About the Database for Results Analysis
- Filtering Data from Results Analysis Database
- Transform Rulesfile

Transformation of Messages

Transformation in Results Analysis is the modification and/or extraction of data from messages. Transformation facilitates data compression by making common the unique segments within a message. Compressing these otherwise unique messages helps the user find root cause failures more quickly.

With regards to data extraction, certain interesting and perhaps important information may be embedded within a message. This data might not be otherwise saved in pre-defined fields within the triage database. Transformation allows you to extract important data into user-defined fields for the purposes of sorting and analysis.

One such example occurs within OVM environment. It is not uncommon for a given message to arise from different parts of the design, or for different reasons. Messages generated from the OVM library have fields such as time, instance, interface identification, and other unique data values which are embedded in the message text. Transforming these messages by extracting the unique bits into separate user-defined fields and removing unique fields from the message itself allows the GUI to group messages according to the extracted field values (for example, grouping messages according to the OVM port from which they originated).

Related Topics

- Transforms
- Transform Rulesfile
- transform and field Example
- Creating a Results Analysis Database (TDB)
- About the Database for Results Analysis

Transforms

A *rulesfile* is a file which contains a collection of transforms. A *transform* is a Tcl construct that you write in order to parse and slice a message in such a way as to reveal only that information which is interesting and/or relevant to the analysis of your verification results.

- Log files require a transform rulesfile.
- WLF files do not require transforms, but use them to greatly improve the analysis of your results in the TDB.
- triage dbfile commands using UCDB files as inputs do not require the use of transforms, although they can be useful. See "Creating a Results Analysis Database (TDB)".

Transforms improve usability, in any case. Although WLF and UCDB files may already contain the necessary information for extraction into msg, time and severity fields, by default, the Results Analysis tool displays all messages whose strings differ only slightly (by time, port number, or design module, etc.) as unique messages. In other words, you could get what could be a potentially staggering number of messages when only a few basic types of messages actually exist.

Related Topics

- Transform Rulesfile for rulesfile syntax
- transform and field Example for examples of transform syntax
- Creating a Results Analysis Database (TDB)
- About the Database for Results Analysis

Triage Command Examples

Example 6-1. A Simple Example

The "triage dbfile" command imports UCDB and WLF database files into a triage database, which has the default name of *questasim.tdb*. If the simulation of testcase 'example1' generated *example1.ucdb* and *example1.wlf*, you could execute this command at the Questa SIM command line:

```
triage dbfile -clear example1.ucdb
```

The -clear option resets the *questasim.tdb*, which is the default name for the TDB, clearing it of any previously imported data. The UCDB data is stored in a test records table, while the WLF data is stored in a message information table. The data in the UCDB file is linked through the WLFNAME attribute pointing to the associated *.wlf* file. The keyword for associating the two subsets of data is the testname, which in this case is “example1”.

Example 6-2. Over-writing Data

By default, the "triage dbfile" does not over-write data. It assumes that importing tests and/or messages under the same testname as previous tests and/or messages — either from the WLF, Log or UCDB file — is not the desired behavior, since that would result in duplicate records in the database. For example, the following command sequence results in an error message that the data has already been loaded in the database:

```
triage dbfile -clear example1.wlf
triage dbfile example1.wlf
```

However, it could be valid to over-write the data if the contents have changed. If, for example, the second *example1.wlf* above has more data because you ran the simulation longer. In a case such as this, use the -force option to over-write the database information:

```
triage dbfile -clear example1.wlf
triage dbfile -force example1.wlf
```

Example 6-3. Appending Data to an Existing Test

You may wish to load additional messages associated with a test for which messages already exist: for example, if some messages to be associated with a given test are contained in the WLF file and other messages from that same test (possibly from a 3rd-party tool) are emitted into a plain-text LOG file. In that case, apply the "-append" switch to the “triage dbfile” command line.

```
triage dbfile example1.wlf
triage dbfile test1.log -append -rulesfile trans.txt
```

Example 6-4. Creating a Database from Many Input Files

One way to easily generate a triage database of many tests is to list all the UCDB files in an "inputs" file. The "inputs" file is an ASCII file listing all the UCDB files, one per line. Relative paths, based upon where the "triage dbfile" command was executed, or full paths are required. For example, if you list all UCDB files generated by regressions in a file named *UCDBinputs.txt*, you would use the following "triage dbfile" command:

```
triage dbfile -clear -inputsfile UCDBinputs.txt
```

The WLF files are automatically imported via the WLFNAME attribute.

The -inputsfile argument can also be used to import multiple WLF and/or LOG files, so long as -testname is not specified on the command line. In this case, the testname needs to be determined from the base name of the WLF/LOG file itself. See "Association of a Testname to WLF or Log Files".

Lockfile

A lockfile is generated before any database writing or reading. Afterwards, the lockfile is automatically deleted. This prevents database corruption if multiple processes or threads want access to one triage database file.

Occasionally, a lockfile will be orphaned due to unusual circumstances, such as a process getting killed while in the middle of an operation. If the lockfile is found to no longer be active, it is automatically deleted. In circumstances where a lockfile exists but should not, you can inspect the lockfile data (pid, hostname, username and date of creation) to confirm that the lockfile is not valid, and manually delete it.

Tip for Grid Use to Avoid Non-deterministic Results

If you want to use some form of parallelism, via grids or other techniques, it is important not to delete or over-write previous database contents. Otherwise, the data produced might be non-deterministic. For example, if you chose to enter a "triage dbfile" command which deleted messages starting with "OVM", the triage database created may contain different "OVM" messages from run to run, depending upon when that deletion command actually executes, which changes depending upon the grid loading.

In order to restore consistency to a database, perform a -clear before launching any grid jobs that may access the database. After doing so, refrain from using "triage dbfile -clear" or "triage dbfile -deletemsg" commands within the grid jobs.

Automatic WLF and Logfile Import Search Order

When the UCDB file that is specified in a triage command contains either a WLFNAME or LOGNAME attribute, the following search order is used to find the WLF or LOG file:

1. Search using the filename given, either a full-path or a relative-path filename
2. Search relative to the RUNCWD attribute in the UCDB file
3. Search relative to the MGC_WD environment variable, if it exists
4. Search relative to the given path of the UCDB file in the "triage dbfile" command
5. Search relative to the ORIGFILENAME attribute in the UCDB file

Search rules 2, 4 and 5 only apply to imports from a UCDB file.

About the Database for Results Analysis

The database (triage database or TDB) used for Results Analysis holds one or more sets of data, where each set corresponds to one unique test. Each test you run in Questa SIM generates a set of data including two subsets of data:

- Test record information — a single collection of data, stored in a UCDB database, and containing information such as username, date and hostname.
- Message information — one or more (usually more) collections of data, stored in a WLF database. The WLF file contains fields/elements such as time, severity, and the actual message string.

Association of a Testname to WLF or Log Files

The testname field, one of the WLF and UCDB predefined fields, is important for linking message and test record information. If the data does not match, either because of mismatched or missing data, then these two data subsets will not get linked when viewing the triage database.

Messages whose testname is blank (i.e.: message imported directly from a WLF or LOG file without the “-testname” command line option) are still visible in text reports and in the GUI, but any per-test fields derived from the UCDB would be blank.

If you are importing WLF or Log files directly (i.e., not via UCDB files and automatic import), there are two ways to determine a testname:

1. via a -testname option
   The user imports one WLF or Log file with one -testname option, and the "triage dbfile" command will write that data into the triage database. If multiple WLF or Log files are listed with one -testname option, an error is generated since it does not make sense to have multiple WLF or Log files linked to one testname.
2. via the filename
   If the user imports *example1.wlf* via a "triage dbfile" command without the -testname option, then the tool uses "example1" as the testname.

Related Topics

- Creating a Results Analysis Database (TDB)
- About the Database for Results Analysis

Filtering Data from Results Analysis Database

Filtering data prior to insertion into the triage database is necessary to prune away low-priority or irrelevant information: you may only want to know about tests which fail with fatal or error conditions. However, the triage database creation step allows you full flexibility to see all data.

You can filter data such as the following:

- Severity - filters data based on severity of the failure message

  Individual messages have corresponding individual severities. As each message is processed, before insertion into the triage database, the severity is checked against what the user selects as a filter. The default severity filter is to only allow messages with F or E severity. If a message's severity does not pass the filter, the message is not inserted into the database. You can override the default severities via the -severity option in the "triage dbfile" command.

- Teststatus - filters data based on the value of the “teststatus” attribute from the UCDB.

  The default teststatus filter for "triage dbfile" is fatal, error, merge error and missing. Messages from tests that have a teststatus of warning or OK (pass) are not by default entered in the database. See the -teststatus argument to the "triage dbfile" command for details on changing the default setting.

- Deleting Messages - filters data based on strings contained in the messages

  It is possible to delete messages saved in the triage database, which provides finer granularity of filtering than the -severity option. For instance, the user wants to see some warning messages but delete others. To accomplish this, the user can permit warning messages into the triage database, but delete warnings he/she is not interested in. Here is an example of how to delete messages in the triage database:

  ```
  triage dbfile -deletemsg "*vsim-3812*" -deletemsg "There is an 'U'|'X'|'W'|'Z'|'-'*"
  ```

  Triage commands use "glob" style matching of strings for messages already saved within the triage database. All messages that match are deleted.

To use the GUI to filter data from an open database, right click in the Results Analysis window and select **Filter...** to open the Filter dialog box, where you can choose filter data.

Related Topics

- "triage dbfile" command for a complete list of filtering options and syntax details
- Filtering Data from Results Analysis Database
- About the Database for Results Analysis

Viewing Results Analysis Data

Triage database information can be viewed using the GUI or text reports created using the "triage report" command.

Viewing Results Analysis Data in the GUI

Prerequisites

- You must have already created a TDB. See "Creating a Results Analysis Database (TDB)".

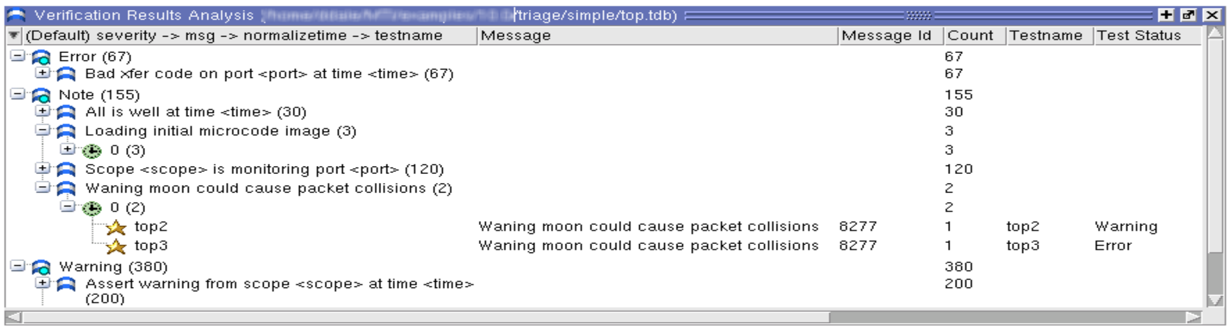
Procedure

1. Open the GUI
2. From the menu bar, select **View** > **Verification Management** > **Results Analysis**.
3. From the menu bar, select **Results Analysis** > **Load Database**. This opens the Load Triage Database dialog box.
4. Enter the name of the database you wish to view, or browse.

Results

The Verification Results Analysis window opens, showing the contents of the specified TDB file.

Figure 6-1. Verification Results Analysis Window



Verification Results Analysis [redacted]triage/simple/top.tdb)

(Default) severity -> msg -> normalizetime -> testname	Message	Message Id	Count	Testname	Test Status
Error (67)			67		
Bad xfer code on port <port> at time <time> (67)			67		
Note (155)			155		
All is well at time <time> (30)			30		
Loading initial microcode image (3)			3		
0 (3)			3		
Scope <scope> is monitoring port <port> (120)			120		
Waning moon could cause packet collisions (2)			2		
0 (2)			2		
top2	Waning moon could cause packet collisions	8277	1	top2	Warning
top3	Waning moon could cause packet collisions	8277	1	top3	Error
Warning (380)			380		
Assert warning from scope <scope> at time <time> (200)			200		

Related Topics

- Viewing Results Analysis Data in Text Reports
- Verification Results Analysis Window

Viewing Results Analysis Data in Text Reports

Once the database has been generated, the "triage report" CLI can print data in various formats and using various filters. The report can either print to the screen to output to a file.

For example, you can create a default formatted text report of a Results Analysis database by entering a command such as:

triage report -name top.tdb

A default formatted report such as the following results. See "triage report" for further details.

Example 6-5. Example Triage Report

```
#
# Total tests                                         1
#
# Total number of tests with fatal or error           0
# Total number of tests with warning                  0
# Total number of tests with pass                     1
# Total number of tests with merge error or missing   0
#
# Message                                            Count    Severity
Category   Time
# -----------                                       ------   --------   -
-------   ---------
# All is well at time <time>                 10      Note      MISC
10 ns
# Loading initial microcode image                     1        Note
MISC       0 ns
# Scope <scope> is monitoring port <port>             40       Note
MISC       1 ns
#
```

Counts and Compression in the Report

In order to provide a useful summary of messages in the report, data has been compressed, as indicated by "count". The count indicates how many same data items have been compressed for the display.

The compression algorithm seeks to compress the message, severity and category data. When it finds a unique set of message, severity and category data, that unique set is saved off, including the time. When it finds additional sets of message, severity and category data matching a previous unique set, then the count is incremented.

In the non-detail formats (see "Report Formats") the data set for compression can be changed if the -concise {T|F} option is invoked. In this case, testname and/or filename are included in the message, severity and category set. For maximum compression, do not use the -concise {T|F} option. Adding testname into the report provides valuable information, but it could be a secondary step in the regression triage process.

Report Formats

There are two sets of data — detailed and non-detailed — but three different report formats:

- Non-detailed Concise —

  A row-and-column format showing message, count, severity, category, time, and possibly testname and filename data. Since the row-and-column format has fixed column widths, strings longer than the width of the column in which they appear are truncated. The testname and filename are not displayed by default. To see this data, the user needs to invoke the -concise {T|F} option. Without the -concise {T|F} option, this report format is the most concise and compressed.

- Non-detailed No-Truncation —

  When data has been truncated in the non-detail concise format such that it is unreadable, the next medium-concise format is the non-detailed, no-truncation format. This format is invoked using the -notruncation option. Here, the row-and-column display is transformed into a line-by-line display where no data is truncated. There is no additional data (to get additional data, see the "Detailed" format) just formatting changes to prevent data truncation.

- Detailed —

  The Detailed format is a line-by-line format that displays relevant message, assertion, test record and user-defined information. Once the initial steps in the triage process are complete (e.g., looking for the highest priority failure(s)), additional information can be useful in debugging and assigning resources. For example, finding an assert message and getting source details on which assert fired.

The non-detail concise and no-truncation formats both show the same data but presented in different formats. The detailed format displays extra data not found in the non-detail formats.

Filtering the Text Report Output

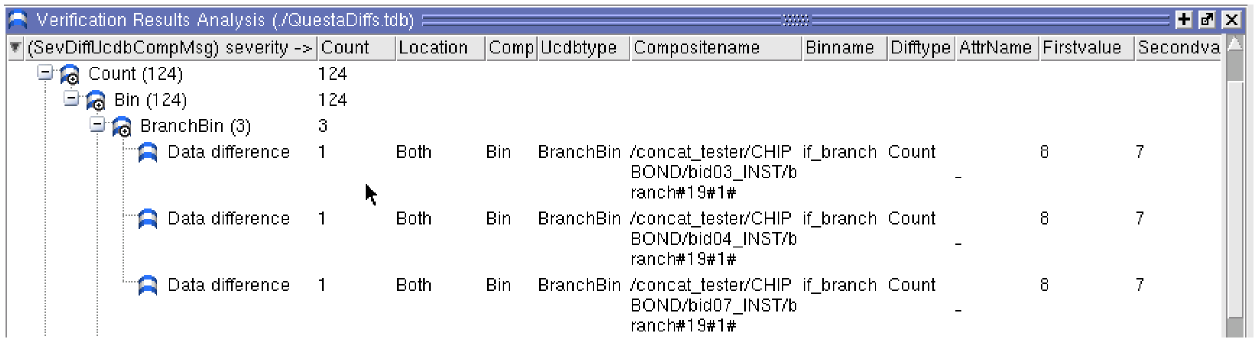
Filtering is performed prior to insertion into the database via the "triage dbfile" command. However, additional filtering can be applied afterward via the “triage report” command to further narrow your search for important information. You can filter reports by severity and teststatus. See the "triage report" command for syntax.

Viewing vcover diff Results in the Results Analysis Window

The Questa SIM vcover diff command creates a report of the component level differences between two UCDBs. The results are written to stdout by default. However, you can open these same results in the GUI by adding the -gui switch to the vcover diff command.

The -gui switch first generates the differences, then transforms them into a triage database (.tdb), and finally, automatically brings up the GUI and displays the results in the Verification Management Results Analysis window.

Figure 6-2. vcover diff Results in the Verification Results Analysis Window



Verification Results Analysis (./QuestaDiffs.tdb)

(SevDiffUcdbCompMsg) severity ->	Count	Location	Comp	Ucdbtype	Compositename	Binname	Difftype	AttrName	Firstvalue	Secondva
Count (124)	124									
Bin (124)	124									
BranchBin (3)	3									
Data difference	1	Both	Bin	BranchBin	/concat_tester/CHIP BOND/bid03_INST/b ranch#19#1#	if_branch	Count	_	8	7
Data difference	1	Both	Bin	BranchBin	/concat_tester/CHIP BOND/bid04_INST/b ranch#19#1#	if_branch	Count	_	8	7
Data difference	1	Both	Bin	BranchBin	/concat_tester/CHIP BOND/bid07_INST/b ranch#19#1#	if_branch	Count	_	8	7

Each of the columns in the window correspond to the fields within the triage database. The information reported is comprised of information such as:

- Type of difference (only present in one or the other, or present but different in both)
- Name and type of component (or parent for flags, attributes, etc). Note that this is exactly the UOR primary key, i.e. the match basis.
- Aspect of difference (scope, bin, attribute, goal, etc)
- Actual difference in values where appropriate
- Counts

For details regarding the information contained in the results of the vcover diff command, see the UCDB API User's Manual.

Predefined Fields in the WLF and UCDB

Two major sets of predefined fields exist: one corresponds with WLF data (Message, listed in Table 6-1) another is matched to the UCDB test record data (Test, listed in Table 6-2).

It is possible for data extracted from the message text to override the values of one or more predefined fields. In the case of messages extracted from plain-text LOG files, this is the only way to get values into the predefined fields. To avoid confusion, do not create a new user-defined field whose name matches an already a predefined field intended for the same type of information.

The following fields hold particular importance:

- Required Severity Field Entry

  All messages that come from WLF files automatically contain a severity. All messages that come from LOG files via transformation must also contain a severity. If there is no recognized severity, the message is not inserted into the triage database.

  The triage tool currently recognizes severities that start with one of these letters, I, F, E, W, or N. Severity strings which start with any letter other than those listed are not inserted into the database. By intentionally setting the severity of any given message to blank, you effectively block its insertion into the database.

- Message Field Recommended

  While it is possible to assign or overwrite the “msg” field to blank, it is not recommended since the message field is usually very useful in results analysis. It is also the primary key in the "triage report" command. Without unique messages, all data will be lumped together in the "triage report" output.

  Also, without message data, a powerful field is lost in the GUI hierarchical view.

- Testname Field

  The "testname" field is the primary key between the set of message data and the test record information (UCDB predefined fields data).

Message Table Predefined Fields

Table 6-1 contains a list of predefined fields from WLF files. The "testname" field is the primary key between this set of data and the test record information (UCDB predefined fields data).

Table 6-1. Predefined Fields from Message Table

Index #	Field
0	primaryid
1	msgid
2	msg
3	idname
4	time
5	iteration
6	category

Table 6-1. Predefined Fields from Message Table

Index #	Field
7	severity
8	objnames
9	process
10	region
11	filename
12	lineno
13	asrtfilename
14	asrtlineno
15	tchkkind
16	asrtstime
17	asrtexpr
18	asrtname
19	fileno
20	verbosity
21	effectivetime
23	testname
23	wlffilename
24	wlftimeunit
25	normalizetime

The "primaryid" field is used as a primary key between message data records and any user-defined field data records. The primaryid field may not be overridden from within a transform.

The index numbers provided in the table can be used with the triage query command.

Test Table Predefined Fields

Another set of predefined fields are UCDB fields. It is important to not use these predefined field names in triage transformations as they may conflict with this set of data. These field values are determined when importing UCDB files and are not overwritten by triage transformations. Index numbers are provided for the use of the triage query command.

Table 6-2. Predefined Fields from Test Table

Index #	Field
0	filename
1	testname
2	teststatus
3	simtime
4	simtimeunits
5	cputime
6	seed
7	testargs
8	vsimargs
9	comment
10	compulsory
11	date
12	userid
13	runcwd
14	hostname
15	hostos
16	wlfname
17	logname

User-defined Fields

Any field specified in a rulesfile matching one of the predefined fields listed above is considered a user-defined field. These user-defined fields can be unique per message since they are specified based upon message matching. As such, they are different from predefined fields because any given user-defined field may be present in some messages while missing in others.

Transform Rulesfile

In order to describe the transformation of message information from an ASCII logfile or WLF file, a rulesfile is required. This ASCII text rulesfile contains the "transform" construct for message matching and a "field" construct for data extraction.

transform and field Syntax

The transformation rulesfile, e.g., *transformation.txt*, contains a set of transform constructs. Each transform construct contains a regular expression for message matching. For data that you want to extract to input into the triage database, surround that data with () to indicate extraction. These () extraction chunks will be used later by in the "field" construct where the first () is mapped to $1, the second () to $2, and so on. Each transform construct can contain an empty field or a set of field constructs. The regular expression syntax matches that used by the “regexp” command in TCL.

```
transform {<regexp>} [-name <str>] {
        <field_construct>
        [<field_construct>]

}
```

where <field_construct> is:

```
field <fieldname> {[$<n>|<str>]+}
```

The <fieldname>s are either user-defined or predefined. If predefined fields are specified, then any previous data is overwritten. See “Predefined Fields in the WLF and UCDB” for lists of predefined fields, and practical advice on when to use them.

If there is no match for a given message within the transform rulesfile, the message is handled according to the type of file from which is originated. In the case of a message extracted from a WLF file, the message is stored as-is in the triage database, subject to any other filtering (such as message severity) which may apply to the message. In the case of a message extracted from a LOG file, the message is dropped.

The reason for dropping LOG file messages which do not match any transforms is that without details such as the time of the message and/or it's severity (both of which must be supplied by transforms since a plain-text LOG file contains no meta-data), the message becomes mere noise in the database. This is also why a transform rulesfile is required to import a plain-text LOG file.

Transform Examples

Example 6-6. transform and field Example

The following is a example of one complete rule within a rulesfile.

```
transform {^(assert) (\S+) (at time) (\d+).*} -name assert_Rule {
    field msg {$1 $3}
    field severity {$2}
    field time {$4}
}
```

Given a logfile with this message:

```
assert warning at time 50
```

The output is:

```
# Original line: 'assert warning at time 50'
# Transformed fields:
#     msg = assert at time
#     severity = warning
#     time = 50
```

Note

If you set the "time" field you should also set the "wlftimeunit" field. If you do not:, (a) for messages from WLF files, the timeunit originally stored with the message will be used, or (b) for LOG files, the time value will be interpreted as nanoseconds.

Example 6-7. Another transform Example

Given a message such as:

```
Error: Bad transaction at 1130 ns on port CPU1 (user1 2008/02/31): missing
ack
```

and the following transformation rule:

```
transform {([^:]+): (.*?) at (\d+ \S+) on port (\S+) [(][^)]*[)]: (.*)$} -
name bad_transaction {
    field msg {$2: $5}
    field severity {$1}
}
```

The final message extractions/modifications should be:

```
msg = Bad transaction: missing ack
severity = Error
```

In this case, the extracted port value could be stored as a user-defined field with the following additional field construct:

```
    field port {$3}
```

This field construct allows you to group and/or sort messages by the identifier string of the port from which they were generated.

Related Topics

- Transformation of Messages
- Transforms
- Transform Debug

Matching Messages Spanning Multiple Lines

To accommodate logfiles that contains messages longer than a single line, include the newline character "\n" in the regular expression in the transform construct. If the longest multi-line message is longer than 10 lines (the default) per buffer, use the -buflines command argument to change the default number of lines.

Example 6-8. Multiple-line Matching Example

Here is an example of a message requiring multiple-lines:

```
Message:    Assert!     Time: 11 ns
    File: Package.sv
    Severity: Error
    Line: 23
```

A transform construct can be written to match this message, as follows:

```
transform {^Message:[ \t]+(Assert!)[ \t]+Time: (\d+) \S+\n[ \t]+File:[
\t]+(\S+)\n[ \t]+Severity:[ \t]+(\S+)\n[ \t]+Line:[ \t]+(\d+)\n} -name
assert_Rule {
    field msg {An $1}
    field time {$2}
    field file {/u/$3}
    field severity {$4}
    field line {line number $5}
}
```

Notice the "\n" characters within the regular expression of the transform construct. These mark the places in the logfile where the message has continued into a new line.

If you put this message and rulesfile into the "triage dbfile" command, the output would be:

```
# Original line: 'Message:  Assert!     Time: 11 ns
#   File: Package.sv
#   Severity: Error
#   Line: 23'
# Transformed fields:
#     msg = An Assert!
#     time = 11
#     file = /u/Package.sv
#     severity = Error
#     line = line number 23
```

Transform Debug

If the transformation process does not yield the specific results you expect, there are several additional levels of debugging which provide visibility to your transforms through the triage dbfile command switches:

- triage dbfile -verbose

  By default, transformation is silent. If you use the "-verbose" switch, something like the following is produced as output:

  ```
  # Original line: 'Bad xfer code on port BOB at time 98'
  # Transformed fields:
  #     msg = Bad xfer code on port <port> at time <time>
  #     category = PortMessage
  #     port = BOB
  #     time = 98
  ```

  "Original line/message" lists each message that was extracted.

  "Transformed fields" lists the fields generated by the transform (if any) that were used to extract the message.

- triage dbfile -nowrite -verbose

  Using "-nowrite" automatically enables the same verbosity as the "-verbose" option. However, if you use both "-nowrite" and "-verbose" together, you get slightly more information:

  ```
  # Original line: 'All is well at time 100'
  # Matching rule: '(.*) at time (\d+)'
  # Matching name: 'atTime' (line 45)
  # Transformed fields:
  #     msg = All is well at time <time>
  #     time = 100
  #     severity = Note
  ```

  In addition to the -verbose level of information, you also get rule that matched the rule's name (if any) and/or the line number where the rule was defined in the rulesfile. If the matching rule has no name, the line number is printed out and the header is "Matching line" instead of "Matching name".

- triage dbfile -debug

  When you use the -debug switch, a third level of debugging information is written to the Transcript pane:

  ```
  # Original line: 'Bad xfer code on port FRED at time 100'
  # Attempt match: 'Delay (\d+) is (.*)'... no match # Attempt match:
  '(.*) from scope (\S+) at time (\d+)'... no match # Matching rule:
  '(.*) on port (\S+) at time (\d+)'
  # Matching name: 'portAt' (line 17)
  # Transformed fields:
  #     msg = Bad xfer code on port <port> at time <time>
  ```

```
#      category = PortMessage
#      port = FRED
#      time = 100
```

In addition the information discussed in the first two levels, you also get a transform-by-transform report of attempted matches until one finally does match, at which point the rule itself, the rule name (if any), and the line number where the rule was defined are printed out.

LOG File Transforms

The examples shown in the section above, Transform Debug, are from WLF file transformations, however LOG file (i.e. plain-text file) transformations output more-or-less the same information, with one exception: for LOG files, the "Original line:" output contains the entire contents of the transform buffer (see The Transform Buffer and Buffer Management). All the lines are shown so that, while debugging, you can figure out where the line breaks fall in the event of a multi-line message.

The Transform Buffer and Buffer Management

When a triage command imports a .log (or plain-text file), it creates a buffer that is checked against all transform constructs within a given rulesfile. If no match is found, the first line is discarded from the buffer and a new line is added from the logfile. Only whole lines are removed.

This process repeats until a match is found. If the matching rule only matched part of a line, the entire line is still removed. Thus, it is important that messages always start on their own line, never on the same line as a previously-matched message. If no matching rule is found on a line, **only** one line is discarded from the buffer and the next line is evaluated.

When a match is found, the command discards the exact number of lines within the buffer as the number of lines in the transform construct regular expression. This repeats until the end-of-file for the logfile has been reached.

Managing Multi-line Messages

Multi-line messages are represented in the rule by a newline ("\n") character in the regular expression. Regular expression rule matches are "anchored" at the start of the buffer. The algorithm for matching does not cross line boundaries to find the start of a message. Even though a matching rule can span multiple lines, in order for it to be considered a match, the first matching character must occur in the first line of the buffer. Otherwise, there could be other messages before the "matching" messages that might be missed if it skipped lines to find the match.

Buffer Flow

Once a match is either found or not, and the matching (or non-matching) line(s) are removed from the buffer, the algorithm proceeds to refill the buffer. If enough lines remain in the LOG file to fill the buffer back up to "N" lines, those lines are read and added to the buffer. Until all the lines in the buffer have been subjected to the transform process, the algorithm continues to attempt to match the buffer against the rules in the rulesfile.

The net effect of this matching is that this "N"-line window which starts at the beginning of the LOG file, advances through the file as it matches and/or discards lines from the file, and it continues until every line has been examined. With every advance, the algorithm attempts to match each rule in turn, and any given rule can actually match multiple lines in the case of a multiple-line message.

Verification Results Analysis Window

The Verification Results Analysis window is used for viewing and sorting message and test data information.

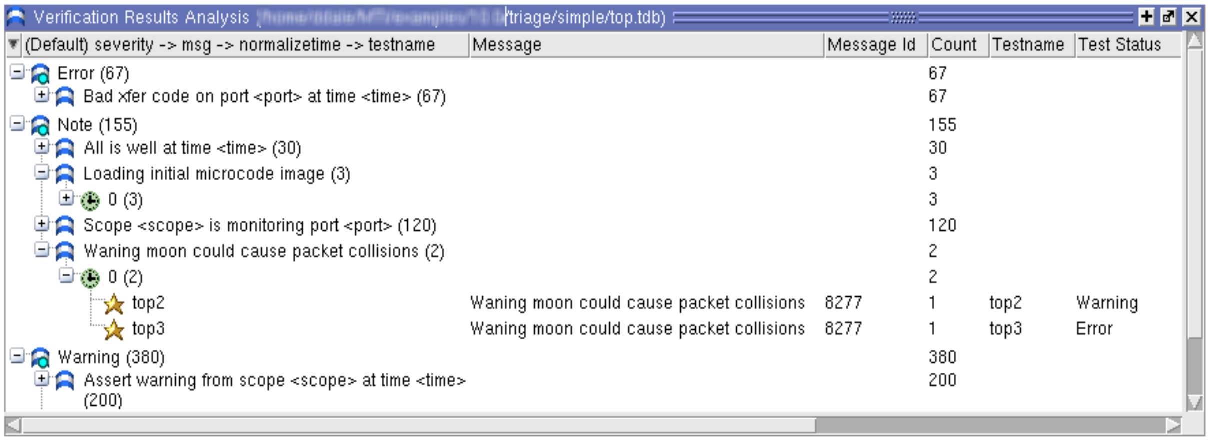
Accessing

Access the window by the following:

- Select **View** > **Verification Management** > **Results Analysis**

When the window is open, you must load the triage database to populate the window. Figure 6-3 shows the Verification Results Analysis window with a triatge database, *top.tdb*, loaded.

Figure 6-3. Results Analysis Tab


Verification Results Analysis ([redacted]triage/simple/top.tdb)

(Default) severity -> msg -> normalizetime -> testname	Message	Message Id	Count	Testname	Test Status
Error (67)			67		
Bad xfer code on port <port> at time <time> (67)			67		
Note (155)			155		
All is well at time <time> (30)			30		
Loading initial microcode image (3)			3		
0 (3)			3		
Scope <scope> is monitoring port <port> (120)			120		
Waning moon could cause packet collisions (2)			2		
0 (2)			2		
top2	Waning moon could cause packet collisions	8277	1	top2	Warning
top3	Waning moon could cause packet collisions	8277	1	top3	Error
Warning (380)			380		
Assert warning from scope <scope> at time <time> (200)			200		

Controlling the Results Analysis Columns

You can customize the appearance of the columns in the Results Analysis window by right-clicking in the column headings and selecting **Select Column Visibility** to display a list of all column headings which allows you to toggle the columns on or off.

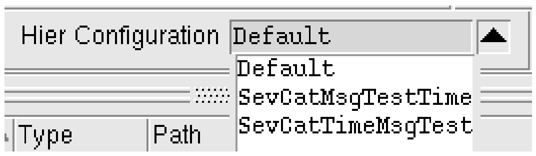
Dragging Information from RA Window

In some cases, it can be helpful to drag and drop contents from a column in the Results Analysis window to the Transcript window. For example, you might begin a command in the Transcript, and drag the instance path, message number, etc. from the Results Analysis window to make the typing task less tedious.

Results Analysis Hierarchy Configuration

The Results Analysis window contains a toolbar used to choose a hierarchy configuration. A “Default” configuration as well as two pre-defined configurations are available.

Figure 6-4. Hierarchy Configuration



Saving Custom Hierarchy Configuration

You can save your own custom column layout and any filter settings to an external file (*triageviewer_hier_config.do*) by selecting **File > Export > Hierarchy Configuration** while the window is active. You can reload these settings with the do command. This export does not retain changes to column width.

GUI Elements of the Results Analysis Window

This section provides an overview of the GUI elements specific to this window.

Menu Items

The following menu items are related to the Verification Results Analysis window:

- **Create Database** — opens the Create Triage Database dialog, the fields in which correspond to the switches and arguments in the triage dbfile command. Refer to that command, as well as the section entitled “Viewing Results Analysis Data in the GUI” for more information.
- **Load Database** — loads the specified TDB into the Results Analysis window.

- **Refresh Database** — recreates the active TDB, using a saved history of previously executed triage dbfile and triage dbinsert commands.
- **Close Database** — closes the active TDB, and removes the displayed data in the Results Analysis window.
- **Hierarchy Configuration** — opens the Hierarchy Configuration dialog, which allows you to save, edit and apply previously saved hierarchy configurations in the Results Analysis window.
- **Import Hierarchy Configuration** — opens the Import Hierarchy Configuration dialog, which allows you to open a previously saved .do file containing a hierarchy configuration.
- **Export Hierarchy Configuration** — opens the Hierarchy Configuration dialog, which allows you to save a TDB database, along with its hierarchy configuration in a *.do* file. The default name is *triageviewer_hier_config.do*.
- **Filter...** — opens the Results Analysis Filter dialog, which allows you to set global and first message filters to apply to message / test data in the Results Analysis window.
    - First Message Filter — Allows you to set whether you display all matching messages into the Results Analysis window, or filter out all but the first message for each unique value of a user-specified WLF file field. You can also select a WLF file field on which to sort the first messages that are displayed.
- **Clear Filter...** — clears the effects of the filter on the TDB.

Related Topics

- Viewing Results Analysis Data in the GUI
- Viewing Results Analysis Data in Text Reports

Chapter 7 Analyzing Trends

Analyzing Trends in Verification

Trends in the coverage achieved over a period of time can be identified and analyzed using a Verification Management feature called “Trend Analysis”.

Objects Available for Trend Analysis

These are the complete set of objects to which trend analysis can apply:

- Verification plan sections
- Design units (module types)
- Design instances
- Covergroups
- Covergroup instances
- Coverpoints and crosses

Usage Flow

1. Run regressions over multiple days/weeks
2. Create a trend UCDB, a merged database from all these regressions
3. Create a trend report to analyze the trends over time

What is a Trend UCDB

The data on which trending analysis relies is captured in a special-purpose UCDB file — referred to as the “trend UCDB” — which represents the set of objects to which trend analysis applies, and the aggregated coverage for all those objects for all inputs.

The trend database (UCDB) differs from other UCDBs in that:

- It captures trendable objects only - i.e., only down to the coverpoint or cross level.
- It preserves all distinct values for those objects based on timestamp of the input. If the same input is given multiple times to merge -trend, it will not have any effect because the redundancy is detected due to the timestamp.

Once you create a trend UCDB, it appears in the **Verification Management > Browser > Trend Analysis** in the GUI. The trend UCDB represents a concise representation of trending data that can be archived for future reference. You can also merge together multiple trend UCDBs to create a new trend UCDB, essentially concatenating them.

From the trend UCDB, you create a trend report, including a 2-dimensional graph for viewing in:

- the GUI in the Trender pane
- HTML report
- XML report
- text format
- an exported format such as CSV

Coverage Computation for Trend Analysis

Coverage computation follows the identical algorithms to that of "Calculation of Total Coverage" such that:

- Covergroups, covergroup instances, coverpoints, and crosses are consistent with the coverage computation of SystemVerilog.
- Verification plan section coverage is a weighted average of subsidiary objects associated with that section: whether that is a number of verification plan sub-sections or linked coverage objects.
- Coverage of a design instance is the weighted average of different kinds of coverage within that instance and all its sub-instances. This includes code coverage.
- Coverage of a design unit is the weighted average of different kinds of coverage after coverage from all instances are merged together. This includes code coverage.

There are globally assignable weights for different kinds of coverage, used for computing overall coverage of design instances and design units.

Creating a Trend UCDB

You can create a Trend database (which is a specially purposed UCDB) using the command line or the GUI. The input UCDBs need not be merged files; they could be individual files. They can also be trend UCDBs themselves.

Command Line

To create a trend UCDB using the command line interface, use a command such as the following:

vcover merge -trend [-output] <trend ucdb> <ucdb inputs>

See vcover merge -trend for details on command syntax.

Tip: The external coverage utility, vcover, is available in both operating system shells and the vsim command (Tcl) shell.

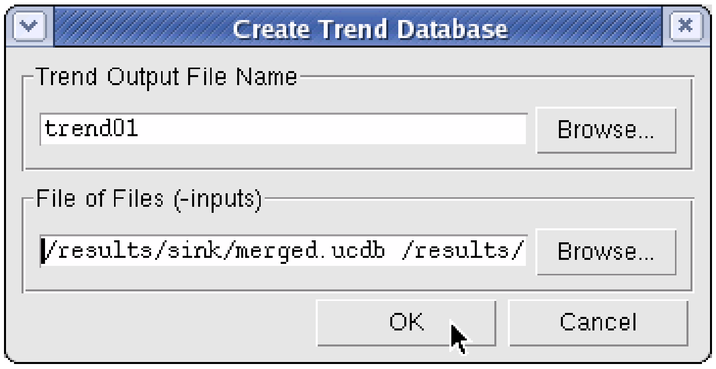
GUI Procedure

To create the database that can be opened in the Trender window:

1. View the UCDB(s) the **Verification Management** > **Browser**.
2. Select the UCDB(s) to be merged into a Trend UCDB, and right click to bring up the menu.
3. Select **Trend Analysis > Create Trend Database**

   The Create Trend Database dialog box opens up in a tab, as shown in Figure 7-1.

Figure 7-1. Create Trend Database Dialog Box



Trend Output File Name: trend01

File of Files (-inputs): /results/sink/merged.ucdb /results/

4. Enter the name of the Trend UCDB file you are creating into the Trend Output File Name field. You can Browse the hierarchy to update one which already exists.
5. Enter the list of UCDBs to use as inputs for the Trend database. You can select multiple files easily by selecting the Browse and highlighting all files to be merged.
6. Select **OK** to create the merged Trend database UCDB.

Results

When the Trend database is successfully produced, a message similar to the following appears in the Transcript window:

```
...
# Merging file /results/sink/mm3.ucdb
# Merging file /results/sink/mm4.ucdb
# Merging file /results/sink/mm5.ucdb
# Writing merged result to trend
```

Use Scenario for Adding Attributes to a Trend UCDB

You can add your own attributes to a Trend database for the purpose of seeing that name/value expressed in the Trend report, using the -trendable argument to the coverage attribute command. These attributes would ultimately be viewable using vcover report -trend.

One potential usage scenario for adding attributes (test, fails, passes) for trending, over a period of time, would be as follows.

1. Run tests. Let's say these are the tests from the first month of testing. Merge the results into single UCDB, "snapshot1" using a command such as the following:

    vcover merge test1.ucdb test2.ucdb [...] test87.ucdb -output snapshot1.ucdb

    Run more tests over the next month and merge these into a second snapshot file:

    vcover merge test88.ucdb test89.ucdb ... test143.ucdb -output snapshot2.ucdb

    Run still more tests and... :

    vcover merge test144.ucdb test145.ucdb ... test202.ucdb -output snapshot3.ucdb

2. Incorporate the desired trendable attributes into those snapshots using commands such as:

    ```
    vsim -c -viewcov snapshot1.ucdb
    coverage attribute -ucdb -name Tests -value 87 -trendable
    coverage attribute -ucdb -name Fails -value 47 -trendable
    coverage attribute -ucdb -name Passes -value 40 -trendable
    coverage save snapshot1.ucdb

    vsim -c -viewcov snapshot2.ucdb
    coverage attribute -ucdb -name Tests -value 98 -trendable
    coverage attribute -ucdb -name Fails -value 58 -trendable
    coverage attribute -ucdb -name Passes -value 40 -trendable
    coverage save snapshot2.ucdb

    vsim -c -viewcov snapshot3.ucdb
    coverage attribute -ucdb -name Tests -value 123 -trendable
    coverage attribute -ucdb -name Fails -value 40 -trendable
    coverage attribute -ucdb -name Passes -value 83 -trendable
    coverage save snapshot3.ucdb
    ```

    The vsim -viewcov command opens the snapshot UCDB; coverage attribute -trendable adds the specified attribute names and values into the database; and coverage save saves the new attributes into the snapshot.

3. Perform a trending merge on the three snapshot UCDBs and see the numbers for Tests, Fails and Passes. The vcover merge -trend command creates trend data for those trendable attributes set on merging tests.

    vcover merge -trend trend.ucdb snapshot1.ucdb snapshot2.ucdb snapshot3.ucdb

For more details on trend reporting, see "Creating and Viewing a Trend Report".

Creating and Viewing a Trend Report

The vcover report -trend command can be used to create a report for viewing the following:

- HTML
- XML
- Text
- Comma Separated Value (CSV) files

The resulting report contains a 2-dimensional graph. See the vcover report for details on syntax and usage.

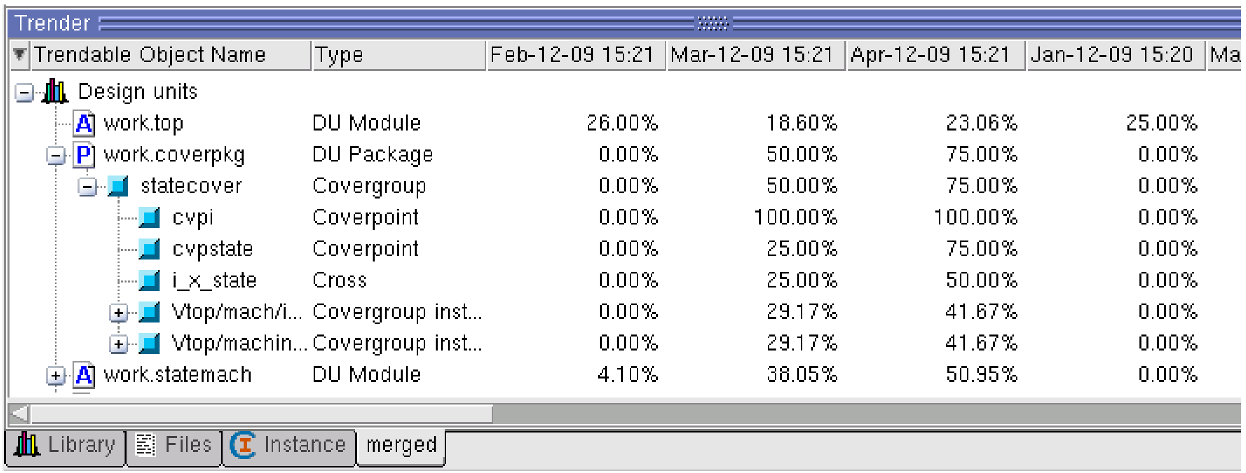
Viewing Data in the Trender Window

To create a report that is viewable in the GUI:

1. View the trend database in the **Verification Management > Browser**.
2. Select a trend UCDB to analyze, and right click to bring up the menu.
3. Select **Trend Analysis > View Trender**

   The Trender window opens up in a tab, as shown in Figure 7-2.

Figure 7-2. Trender Window



Trendable Object Name	Type	Feb-12-09 15:21	Mar-12-09 15:21	Apr-12-09 15:21	Jan-12-09 15:20	Ma
Design units						
work.top	DU Module	26.00%	18.60%	23.06%	25.00%	
work.coverpkg	DU Package	0.00%	50.00%	75.00%	0.00%	
statecover	Covergroup	0.00%	50.00%	75.00%	0.00%	
cvpi	Coverpoint	0.00%	100.00%	100.00%	0.00%	
cvpstate	Coverpoint	0.00%	25.00%	75.00%	0.00%	
i_x_state	Cross	0.00%	25.00%	50.00%	0.00%	
Vtop/mach/i...	Covergroup inst...	0.00%	29.17%	41.67%	0.00%	
Vtop/machin...	Covergroup inst...	0.00%	29.17%	41.67%	0.00%	
work.statemach	DU Module	4.10%	38.05%	50.95%	0.00%	

4. Open the graph from the Trender window:

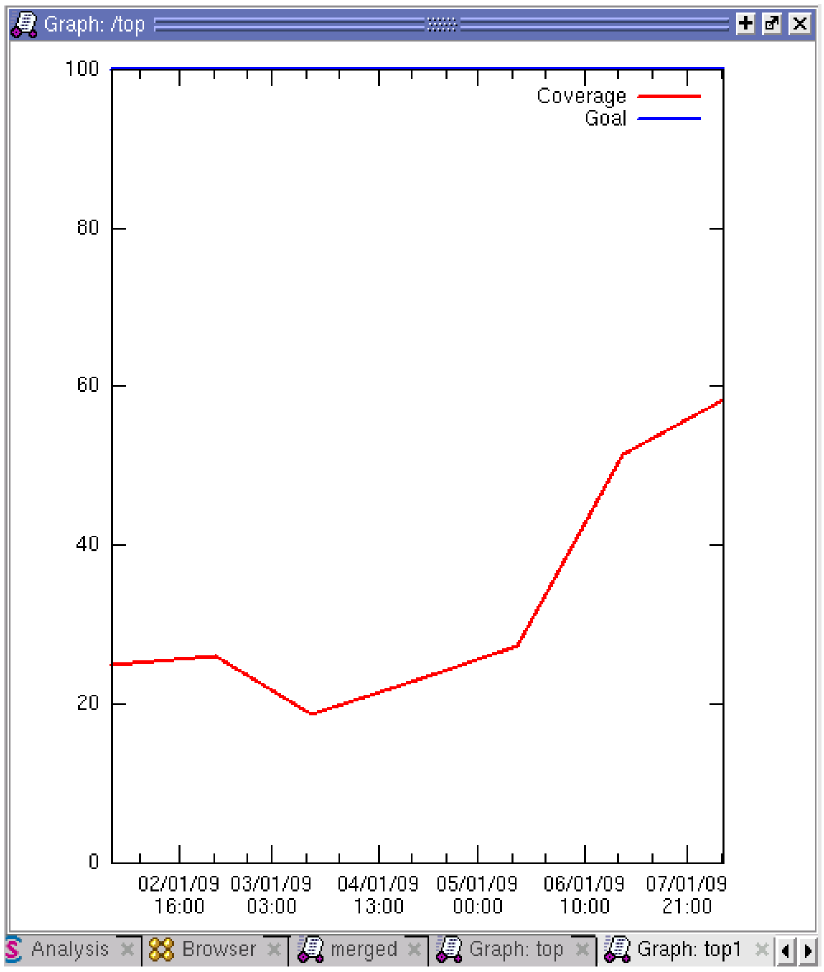
   a. Select the object whose coverage over time you wish to examine.

   b. Right click and select:

- **View Trend Graph Using Local Coverage** — opens graph with local aggregation.
- **View Trend Graph Using Recursive Coverage** — opens graph with local aggregation.
- **Select Coverage and View Graph...** — opens a dialog box for enabling coverage types and recursive aggregation for viewing in the graph.

When one of the above selections is made, A 2-dimensional graph opens up in the Verification Management area, similar to that shown in Figure 7-3.

Figure 7-3. Trend Graph



Appendix A
xml2ucdb.ini Configuration File

Note

The functionality described in this chapter requires an additional license feature for ModelSim SE. Refer to the section "License Feature Names" in the Installation and Licensing Guide for more information or contact your Mentor Graphics sales representative.

Overview

The *xml2ucdb.ini* configuration file, as shipped with the tool, is sufficient for the vast majority of all imports. Generally, you should not need to modify this file to import a verification plan if the verification plan (a top level testplan) was created:

- using one of the tools listed in “Supported Plan Formats”, and
- following the guidelines listed for that particular format.

However, verification plans written outside the scope of these guidelines may require you to customize the file’s extraction parameters.

This appendix contains the file conventions and syntax details for the *xml2ucdb.ini* file. The parameters contained in this file govern the extraction of data during the import of a verification plan. The format and syntax details are provided to assist you in customizing them for your use.

File Location

<installDir>/vm_src/ directory

XML Version Support

The xml2ucdb utility and Verification Plan Import Questa SIM uses to convert the XML file to UCDB format accepts XML version 1.0 files. For details on XML, see the “Extensible Markup Language (XML) 1.0 Specification,” available on the web.

File Conventions

- Plurals — Parameters whose name is plural can accept multiple characters, strings, tag names, and so forth, in a single value argument.
    - Parameters ending with *tag expect a single XML tag name.

    - Parameters ending with *tags expect a list of one or more XML tag names (for example: a [taglist]).

- Comments — a comment begins with a semicolon, and ends at the end of the line containing the semicolon.
- Pseudo-XPath syntax — Each tag in a taglist can include a subset of the XPath syntax to identify elements not only by tag name, but also by the contents of attributes attached to said elements. The following caveats apply to Questa SIM's subset, pseudo version of Xpath syntax:
    - can only handle "=" and "!="
    - can only examine the attribute values attached to the element being compared
    - can only perform one attribute comparison. For example, the following extraction parameter:

      "-starttags Worksheet[@ss:Name=Sheet1]"

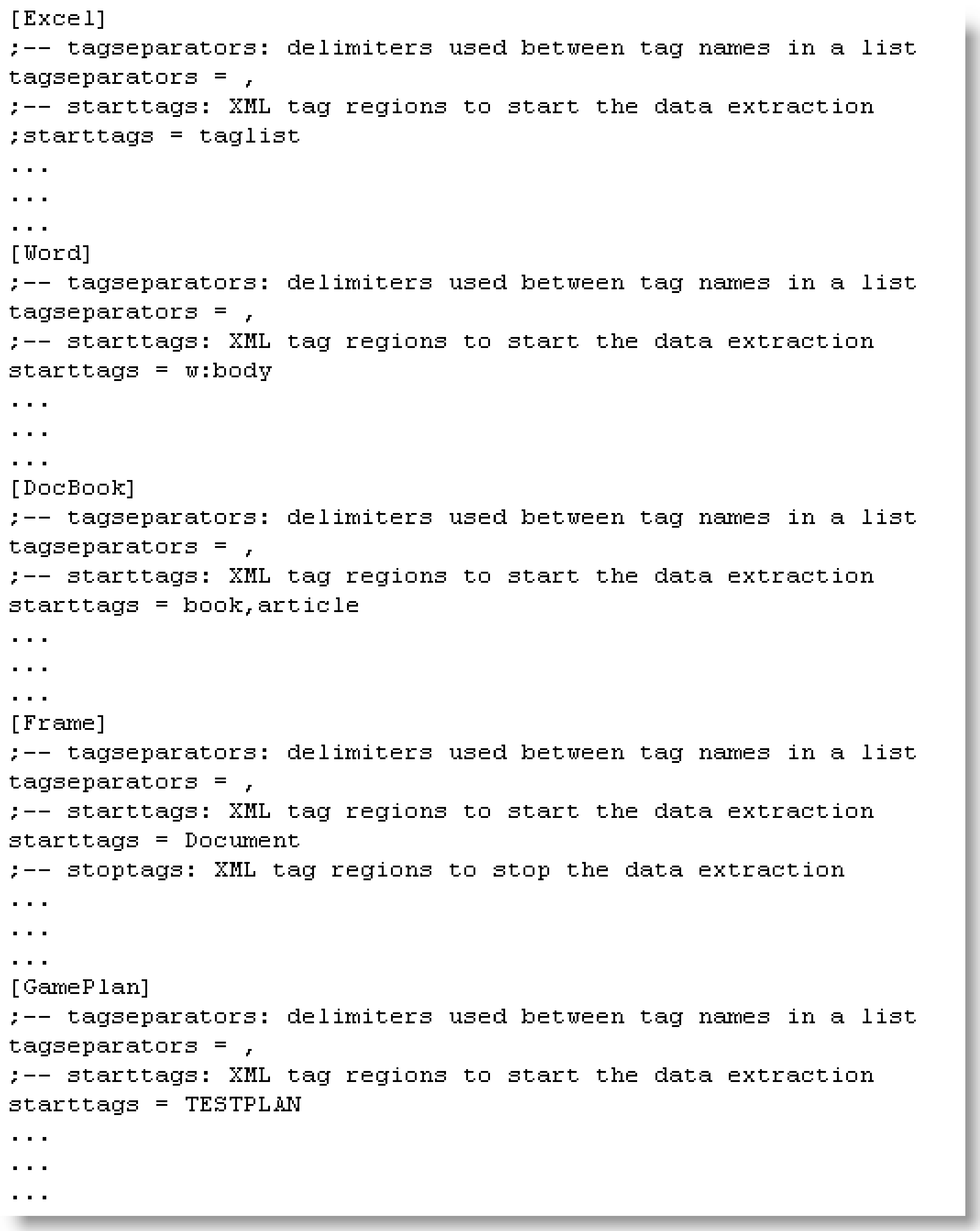
      matches the following element in the incoming XML:

      <Worksheet ss:Name="Sheet1">...</Worksheet>

      but does not match the following element:

      <Worksheet ss:Name="Sheet2">...</Worksheet>
- In the case of a [taglist] parameter value, see the section on the tagseparator parameter in "Parameters Controlling Configuration File Format" for further explanation.

Figure A-1. Default xml2ucdb.ini Sample



```
[Excel]
;-- tagseparators: delimiters used between tag names in a list
tagseparators = ,
;-- starttags: XML tag regions to start the data extraction
;starttags = taglist
...
...
...
[Word]
;-- tagseparators: delimiters used between tag names in a list
tagseparators = ,
;-- starttags: XML tag regions to start the data extraction
starttags = w:body
...
...
...
[DocBook]
;-- tagseparators: delimiters used between tag names in a list
tagseparators = ,
;-- starttags: XML tag regions to start the data extraction
starttags = book,article
...
...
...
[Frame]
;-- tagseparators: delimiters used between tag names in a list
tagseparators = ,
;-- starttags: XML tag regions to start the data extraction
starttags = Document
;-- stoptags: XML tag regions to stop the data extraction
...
...
...
[GamePlan]
;-- tagseparators: delimiters used between tag names in a list
tagseparators = ,
;-- starttags: XML tag regions to start the data extraction
starttags = TESTPLAN
...
...
...
```

File Format

{<section>}

{<section>}

...

where each <section> is:

```
{ <section_heading>
[<descriptive_comment>]
{<setting>}
[<descriptive_comment>]
{<setting>}
...
```

and:

<section_heading> =

[Word] | [Excel] | [DocBook] | [Frame] | [GamePlan]

Required. Defines the program used to create this file. The bracket surrounding the name of the program is required.

<descriptive_comment> =

; <comment describing setting>

Optional. For readability, it is recommended that each setting be introduced by a descriptive comment (begun with a semicolon ";"), which describes the purpose of setting.

<setting> =

<name> = <value>;

Required. This is where you set the value for each of the parameters used by the XML2UCDB in extracting the data from your plan.

The categories for each type of parameter are the following; each of the available parameters is discussed in detail in these sections:

- "Parameters Controlling Configuration File Format" on page 144
- "Parameters Controlling the Inclusion of Data" on page 144
- "Parameters Identifying Section or Column Boundaries" on page 145
- "Parameters Mapping Testplan Data Items" on page 146
- "Parameters Determining Hierarchy and Section Numbering" on page 151
- "Parameters Adding Tag-based Prefixes to Section Numbers" on page 153
- "Parameters Specifying UCDB Details" on page 153
- "Parameters Specifying a Pre-Process XSL Transformation" on page 154

varfile Setting for Parameterizing Testplans

Quite apart from the parameters discussed in the remainder of this appendix is the notion of parameterizing testplan values. You can override any parameter within a testplan with values specified by variables in one of three ways:

- using the xml2ucdb command's -G argument, and/or
- by storing variables in a <varfile> file, which are then referenced using the xml2ucdb command's -varfile argument
- within the "varfile" setting in the *xml2ucdb.ini* file (or like purposed <user_named>.ini file)

To specify the use of such overrides from the .ini file, use the varfile setting.

In order for the varfile setting to function properly, it must be placed in the section of the file for the format you specify using the "format" parameter ("[Word]", "[Excel]", etc.) in order to function (see "Parameters Controlling Configuration File Format").

Syntax

varfile = <relative_path>/<file>;

where:

<relative_path> is relative to the directory in which the xml2ucdb command is issued.

<file> contains a list of "variable=value" strings, one per line, such as:

```
var1=val1
var2=val2
var3=val3
```

Related Topics

Parameterizing Testplans	Guidelines for Writing Verification Plans
Importing an XML Verification Plan	xml2ucdb command

Parameters Customizing Verification Plan Import

Parameters are available to configure the importation of customized verification plans. A customized verification plan is one which, when originally written, contained deviations from the suggestions in "Guidelines for Writing Verification Plans". The parameters listed in this section are used to control how the data is extracted during the importation process.

Parameters Controlling Configuration File Format

The following parameters have to do with locating and/or interpreting the extraction parameters themselves.

Table A-1. Location and Extraction Meta-parameters

Parameter	Value	Command Line	Description
format	<string>	-format	Name of section in INI file containing base parameters (command-line only)
tagseparators	<chars>	-tagseparators	Delimiter for separating tag names in options that accept list of tags as value. Default value is a comma (“,”)

The “format” parameter enables the reading of base parameter values from the configuration (xml2ucdb.ini) file. The configuration file is divided into named sections, one for each XML format supported by the import utility. If the "-format=” option is specified in the xml2ucdb command, the named section is read in prior to processing other command-line arguments. The effect of this is that the parameters in the configuration file serve as a base, on top of which the command-line options serve as overrides, allowing the user to fine-tune the extraction process.

The “tagseparator” parameter specified the character (or characters) that will be recognized as delimiters when parsing <chars> parameter arguments. A <chars> parameter consists of one or more XML tag names separated by one of the tagseparators characters. For example, if the tagseparators parameter was set to "@", then the [taglist] string "Fred@Bill@Bob" would refer to three XML tag names: Fred, Bill, and Bob. Spaces are ignored in a taglist -- however, if the value is passed on the command line, proper attention to the delimiter and quoting rules of the command shell in use is advised.

The command line and the configuration file have independent tagseparators settings. This enables you to override parameters requiring a list of tags without knowing (or conforming to) the tagseparators setting used in the configuration file. In addition, the setting of the tagseparators parameter in the configuration file affects only those parameters which come after it in the file. Thus, you can use multiple tagseparators settings in a single configuration file (however, that means the sequence of the parameters in the file is important).

Parameters Controlling the Inclusion of Data

The following parameters allow the extraction process to focus on a particular segment of the XML input and to exclude uninteresting XML data. Capturing is either enabled or disabled at any given time. The default depends on the type of document we are processing (see below for a more detailed discussion of document types).

Table A-2. Parameters for ID of Item Containing Testplan

Parameter	Value	Command Line	Description
starttags	[taglist]	-starttags	Tag(s) which initiate data capture
stoptags	[taglist]	-stoptags	Tag(s) which terminate data capture
excludetags	[taglist]	-excludetags	Tag(s) to be excluded from data capture
startstoring	[string]	-startstoring	Contents of "Section" field which initiates data capture

It is also possible to trigger capture based on the contents of a field, using the startstoring parameter. This would be used in the case where some number of irrelevant data sections might precede the actual start of the testplan.

Note

The startstoring parameter is only used if auto-numbering is not enabled.

Parameters Identifying Section or Column Boundaries

These parameters allow us to divide the XML input into testplan sections and, within each section, into data items (or columns).

Table A-3. Parameters for ID of Testplan Section and Column Boundaries

Parameter	Value	Command Line	Description
sectiontags	[taglist]	-sectiontags	Tag(s) which start a new testplan section
datatags	[taglist]	-datatags	Tag(s) which start a new data column within a testplan section. Default value is “para”

- At least one sectiontags tag must be defined in order for data extraction to occur. If one or more datatags tags are defined, these tags are used to delimit the data items of each testplan section.
- If no datatags tags are defined, each contiguous string of characters is treated as a data item. This mode, however, should only be relied upon if all fields are explicitly labeled (see Parameters for Associating with Specific Tags), as text regions in the source document may be arbitrarily divided during the XML export process.

Tip: Mutual Exclusions
Use of the exclude and sectiontags tag lists should be mutually exclusive. If a sectionitem tag appears in the exclude list, it does not suspend data capture.

The sectiontags and datatags tag lists are, by definition, mutually exclusive. If a tag appears in both lists, it is treated as a section tag.

Parameters Mapping Testplan Data Items

There are three methods by which data items extracted from the XML file can be assigned to specific "columns" in the testplan (a "column" being a predefined data classification, such as Weight, Goal, Section number, and so forth).

- Parameters for Associating with Specific Tags
- Parameter for Mapping by Column Sequence
- Parameter for Mapping by Explicit Label

The method used depends mostly on the format of the data. In addition, two of the three methods support user-defined columns (which are stored as attributes attached to the testplan scope in the UCDB). The following sections discuss the data item mapping methods in more detail.

Parameters for Associating with Specific Tags

In some XML formats, certain data items are marked with explicit semantic tags. For example, if the documentation tool has a specific place for the "title" of a given section (as in the DocBook format), the user can annotate the testplan section title as a "title", causing the section titles to be annotated with the <title> tag. The XML import utility then maps this directly to the testplan section name field in the UCDB.

The following parameters identify markup tags which indicate data intended for specific fields in the database.

Table A-4. Parameters for Mapping by Tag Association

Parameter	Value	Command Line	Description
titletag	[taglist]	-titletag	Tag(s) which mark the "Title" field of a section
descriptiontag	[taglist]	-descriptiontag	Tag(s) which mark the "Description" field of a section
goaltag	[taglist]	-goaltag	Tag(s) which mark the "Goal" field of a section

Table A-4. Parameters for Mapping by Tag Association

Parameter	Value	Command Line	Description
weighttag	[taglist]	-weighttag	Tag(s) which mark the "Weight" field of a section
linktag	[taglist]	-linktag	Tag(s) which mark the "Link" (aka: "Tags") field of a section (can be multiple)

- For all but the "linktag" element, the data value in question is assumed to be in the element content.
- The linktag is special in that a "link" requires two pieces of information: the name or path associated with the cover item to be linked to the testplan section in question, and the "type" of cover item being linked. The type string must occur as the value of a named attribute (unless there is a labeled "TYPE" data item elsewhere in the section).
- The cover item name/path is either extracted from the content of the linktag element or from a named attribute on the element's start tag.

Parameters Mapping by Attribute Name

If either of the following two parameters are specified, the associated data value will be extracted from the named attribute matching the value of the given extraction parameter.

Table A-5. Data Value Extracted from Named Attribute

Parameter	Value	Command Line	Description
typeattr	[name]	-typeattr	Attribute containing the "type" of each cover item
linkattr	[name]	-linkattr	Attribute containing the "path" or "name" of each cover item

Parameter for Mapping by Column Sequence

The parameter responsible for mapping testplan information to the generated UCDB by column sequence is the **datafields** parameter. The datafields parameter is used to extract data items which always appear in a known sequence in the input file — columns of a spreadsheet, for example. See "Columns in the Plan" for related usage information on testplan contents and mapping.

The datafields parameter uses a set of pre-defined keywords, which are treated by Questa SIM as a kind of "reserved" word. These keywords must be used, and must not be altered.

Requirement: You must specify the predefined keywords in the order that their corresponding columns are found in the Verification plan to be imported. In other words, all predefined keywords in the datafields parameter — and their corresponding columns in the plan — must be present in the specified order. Otherwise, the expected mapping will not occur correctly.

The Column Name in Table A-6 lists these special keywords.

Table A-6. Predefined Column Label (Reserved) -datafield Keywords

Label (for Word formats)	Column Name (for spreadsheet formats)	Description
SECTION	Section	Verification plan section number
TITLE	Title	Title of plan
DESCRIPTION	Description	Text description
LINK	Link	Coverage items in the design, can include path
TYPE	Type	Type of coverage items in the design.
PATH	Path	Path of linked item, if not specified in Link field: Ignored if “-”
WEIGHT	Weight	Weight
GOAL	Goal	Coverage goal of plan item
ATLEAST	AtLeast	AtLeast value; to override the at-least value of a coverage object from testplan. See “Overriding at_least Values in Testplan” for details.
LINKWEIGHT	LinkWeight	Weight for specific link; to override the general weight for the coverage object. See “When Should Items be Excluded from a Testplan?” for details.
UNIMPLEMENTED	Unimplemented	Used to indicate whether testplan object is linked to an unimplemented coverage object. See “Weighting of Unimplemented Testplan Sections” and “Counting Coverage Contribution from Unimplemented Links” for details.

Note

‘AtLeast’, ‘LinkWeight’ and ‘Unimplemented’ are present in the list of known names (i.e. reserved keywords), but need to be added as the last fields in the datafields entry. The exception to this rule is that ‘AtLeast’ is recognized for automatically for Word plans, and is added to the UCDB as the last column (by default).

Syntax rules for the datafields parameter:

- The datafields parameter must have one entry for each sequential, non-blank column in the XML input stream.
- Entries are delimited by one of the tagseparators characters.
- Each entry can be a predefined field name (see Table A-6), a user-defined field name, or a null marker ("-") to skip the column.
- For user-defined fields — the text found in the data item for that column is added to the UCDB as an attribute, and the datafields entry is used as the attribute keyword.

Example A-1. Adding User Defined datafields to the xml2ucdb.ini Configuration File

Add your own field (column) to the testplan UCDB using your own parameters, as shown in the following datafields parameter entry:

```
datafields = Section,Title,Description,OWNER,-,Link,Type,Weight,Goal
```

Notice that the required fields are all present, and that the user added an additional column of data to the plan format: OWNER.

The columns that are read from the XML input stream would be:

1. Section - The section number of the plan section
2. Title — The title (scope name) of the plan section
3. Description — A text description of the plan section
4. OWNER — The "owner" of the plan section (user-defined, stored as an OWNER attribute attached to the plan scope)
5. - — Represents a column of data in the original testplan that is "skipped" (ignored by the XML import utility)
6. Link — The coverage items to which this plan section is linked
7. Type —The type of each coverage item in the previous column
8. Weight — The weight of this plan section
9. Goal — The coverage goal of this plan section

Note that the “AtLeast” parameter is automatically added for input files from Word, but in other formats, it must be added manually to the *xml2ucdb.ini* file, similar to the addition of a user defined parameter (such as “OWNER” in this example).

If the datafields parameter is not set, no "per-column" field assignment is done. The data items must then either appear under a field-specific tag, or be labeled as described in "Parameter for Mapping by Explicit Label".

Parameter for Mapping by Explicit Label

Labeled data items are used in the case where the division of data items is not consistent and/or the sequence of the data in each of the sections is not fixed -- as might be the case for a free-form word processor document. The parameter is **datalabels**. It can also be applied with the xml2ucdb command through the -datalabels argument.

Syntax rules:

- The label keyword must appear at the very start of the data item text content and must be followed by a colon (no spaces).
- The label keyword may be one of the pre-defined keywords (see Table A-6 on page 148), or a user-defined keyword.
- For user-defined fields, you must define the keyword (and optional UCDB keyword) in the datalabels parameter. The datalabels parameter accepts one or more label entries.
- The order of the entries is not important.
- Each entry begins with a label string, usually upper case, with which any data item corresponding to that data column is prefixed in the document text.
- Following the label is an optional field name, separated from the label string with a colon (note that using the colon in this way precludes using the colon as a tag delimiter in the tagseparators parameter). If the field name is supplied, that name will be used as the attribute keyword in the UCDB database. Otherwise, the label string will be used.
- In auto-number mode, a fixed set of pre-defined keywords may also be used. This is primarily for back-compatibility with older configuration files. These labels may be overridden by the datalabels parameter.
- Unlabeled data items which follow a labeled data item are considered part of the previous labeled data item.

Note

The section number and section title cannot be specified using the explicit label method. See "Parameters Determining Hierarchy and Section Numbering".

Case Sensitivity and Field Mapping and Attribute Names

- All pre-defined field names (see Table A-6), with the exception of the Description keyword, used in the datafields and/or datalabels attributes are case-insensitive. In other words, whether the "Section" data field is listed as "Section", "SECTION", or

"SeCtIoN" in the extraction parameter, that field is mapped to the correct place in the UCDB database.

- The Description field, as well as any associated attributes, are stored in the UCDB database as attributes that are attached to the associated verification plan scope. Attribute keywords names are case sensitive, and are determined by the extraction parameter which sets them. For example, if the datafields parameter is set to:

  Section,Title,Description,....

  the section description is stored under an attribute named "Description". Whereas, if the datafields parameter is set to:

  Section,Title,DESCRIPTION,....

  the attribute containing the section description is named "DESCRIPTION". This rule holds true for all user-defined parameters as well.

- The "Description" and all user-defined field names are used as-is and the other pre-defined fields are detected by a case-insensitive match and the corresponding values are stored in their respective (pre-defined) places in the UCDB file.

These mapping rules have several implications:

1. You cannot import a user-defined parameter whose name differs from the name of a pre-defined parameter in case alone. For example, one cannot map an incoming data item into a user-defined parameter named "SeCtIoN", as this field name will automatically map this new data item to the same (pre-defined) plan section number field.

2. It is possible that two verification plans created with different extraction parameters could have inconsistent naming conventions. For example, one plan could save plan section descriptions in a "Description" attribute while another could save the same data item in a "DESCRIPTION" attribute. Caution should be exercised in this regard, as downstream tools (that is, report generators or analysis commands) are likely to respond to the exact case of attribute keywords.

3. Both xml2ucdb and the UCDB API uses a number of fixed attribute keywords for internal purposes (for example: "TAGCMD" or "MERGELEVEL"). There is currently no way to guarantee that the names of the user-defined attributes imported from an XML file do not conflict with these fixed attribute keywords.

Parameters Determining Hierarchy and Section Numbering

There are two basic document styles supported by the XML import utility:

- spreadsheet-like (no auto-numbering)

  Spreadsheet-like documents are flat -- there is no inherent hierarchy in the plan sections. Each plan section is a row in the spreadsheet and each data item is a column in that row.

The data items always appear in a given sequence. Since there is no inherent hierarchy, the hierarchical structure of the plan is represented by a "section number" column which contains outline-like section numbers (for example, "1", "1.1", "1.2", "1.2.1", and so forth) which indicate the plan topology. Auto-numbering is disabled for spreadsheet-like documents.

- word processor-like (auto-numbering)

  Word processor-like documents are hierarchical -- that is, a section may contain zero or more sub-sections. Moreover, since there is no concept of a "column", the data items may appear in any sequence within the section. Because the hierarchy information can be inferred from the document structure, section numbers are not needed. Auto-numbering is enabled so the section number data field is not used (and is typically not included in the document).

Table A-7. Parameters for Hierarchy and Numbering

Parameter	Value	Command Line	Description
autonumber	[0/1]	-autonumber	Switch to enable auto-numbering (1=enabled). Default is 1.
noautonumber	[none]	-noautonumber	Switch to disable auto-numbering (command line only).
startsection	[string]	-startsection	String used to start each series of sub-sections.

- If auto-numbering is enabled, data capture is enabled from the beginning of the document (that is, all document markup tags which match the sectionitem parameter are assumed to represent real testplan sections).
- If auto-numbering is not enabled, data capture must be enabled by a markup tag matching the start parameter or by a section whose "section number" field matches the startstoring parameter.
- The autonumber parameter is not consistent between the command-line and the configuration file.
  - In the configuration file, the autonumber parameter is set to 0 or 1, depending on whether auto-numbering should be enabled.
  - On the command-line, the -autonumber option is a valueless switch that enables auto-numbering (which is *off by default).

  To override auto-numbering settings, use the “autonumber=” argument in the *xml2ucdb.ini* configuration file.
- When auto-numbering is enabled, the "section number" for each section is determined after the data extraction process. The "Section" keyword should not appear in the

datafield parameter. If it does, whatever XML data is mapped to that field will be overwritten by the auto-numbering mechanism.

Parameters Adding Tag-based Prefixes to Section Numbers

In auto-number mode, the section numbers can be prefixed with strings to indicate the type of section found at each level of hierarchy. For example, in the case of Jasper's GamePlan, the "Plan", "Feature", and "Property" Tags are all considered sections of the testplan. Property 3 of Feature 2 of Plan 1 might be numbered "PL1.F2.P3" in the testplan source file. In order to replicate that in the UCDB without resorting to user-defined section numbers, the sectionprefix parameter may be used. If defined, this parameter may consist of a list of strings, separated by tagseparator characters, corresponding to the tags listed in the sectiontags parameter.

Table A-8. Parameters for Designating a Stylesheet: -sectionprefix

Parameter	Value	Command Line	Description
sectionprefix	[string]	-sectionprefix	Strings to be used as section prefixes in auto-number mode

Each string listed in the sectionprefix parameter corresponds to a tag listed in the sectiontags taglist, in the order given. If there are fewer strings listed in the sectionprefix parameter than there are tags in the sectiontags parameter, sections represented by a tag for which there is no corresponding prefix string will not be given a prefix. If there are more strings in the sectionprefix parameter than tags in the sectiontags parameter, the extra prefix strings will be ignored.

Parameters Specifying UCDB Details

The "root" parameter defines the section number to be used for the root scope of the testplan. The "title" parameter defines the section name to be used for the root scope of the testplan.

The "tagprefix" is prepended onto every tag string used in the database. If a tagprefix is not specified, the root of the testplan is used. If neither parameter is specified, the basename (filename with no path and no extension) of the XML input file is used. The tagprefix may only be specified on the command line.

Table A-9. Parameters for Hierarchy and Numbering - UCDB Details

Syntax	Command Line	Description
root [string]	-root	Sets the section number of the root node of the testplan.
title [string]	-title	Sets the title of the root node of the testplan.

Table A-9. Parameters for Hierarchy and Numbering - UCDB Details

Syntax	Command Line	Description
tagprefix [string]	-tagprefix	Prefix used to “unique-ify” the UCDB tag strings.

Parameters Specifying a Pre-Process XSL Transformation

In some cases, the incoming XML format is too complex for the simple tag-based extraction algorithm. This might include cases where the same tag is used for different purposes, depending on context to differentiate one from another, or when the same tag is used for multiple data items differentiated only by the value of an arbitrary attribute. In these cases, it may be necessary to use an Extensible Style Language (XSL) transformation stylesheet to convert the incoming XML file to a form from which the xml2ucdb extraction algorithm can identify and extract the data. If the stylesheet parameter is set, xml2ucdb will invoke an XSLT engine to convert the incoming XML file, using the designated stylesheet to guide the conversion.

Table A-10. Parameters for Designating a Stylesheet

Parameter	Value	Command Line	Description
stylesheet	[filename]	-stylesheet	Designates an XSL stylesheet used to transform the incoming XML file

The transformation engine used is an open-source utility called “xsltproc”. This utility has not yet been approved for release with Questa SIM, so you must download this utility and ensure it appears on the search path.

Index

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. **ORDERS, FEES AND PAYMENT.**

1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.

1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.

1.3. All Products are delivered FCA factory (Incoterms 2000), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer requests any change or enhancement to Software, whether in the course of receiving support or consulting services, evaluating Software, performing beta testing or otherwise, any inventions, product

improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

4. **BETA CODE.**

   4.1. Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.

   4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

   4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. **RESTRICTIONS ON USE.**

   5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use it except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Software or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Software, or disclose to any third party the results of, or information pertaining to, any benchmark.

   5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.

   5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at http://supportnet.mentor.com/about/legal/.

7. **AUTOMATIC CHECK FOR UPDATES; PRIVACY.** Technological measures in Software may communicate with servers of Mentor Graphics or its contractors for the purpose of checking for and notifying the user of updates and to ensure that the Software in use is licensed in compliance with this Agreement. Mentor Graphics will not collect any personally identifiable data in this process and will not disclose any data collected to any third party without the prior written consent of Customer, except to Mentor Graphics' outside attorneys or as may be required by a court of competent jurisdiction.

8. **LIMITED WARRANTY.**

8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

9. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10. THE PROVISIONS OF THIS SECTION 11 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

12. **INFRINGEMENT.**

12.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

12.2. If a claim is made under Subsection 12.1 Mentor Graphics may, at its option and expense, (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.

12.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

12.4. THIS SECTION 12 IS SUBJECT TO SECTION 9 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS FOR DEFENSE, SETTLEMENT AND DAMAGES, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

13. **TERMINATION AND EFFECT OF TERMINATION.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term.

13.1. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.

13.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.

14. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products and information about the products to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.

15. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.

16. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.

17. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FLEXlm or FLEXnet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 17 shall survive the termination of this Agreement.

18. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the United States. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, USA, if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. This section shall not

restrict Mentor Graphics' right to bring an action against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

19. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

20. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing by authorized representatives of the parties. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 100615, Part No. 246066