

Questa[®] Verification Management User's Manual

Including Support for Questa SV/AFV

Software Version 2020.4

Document Revision 5.3

Unpublished work. © Siemens 2020

This document contains information that is confidential and proprietary to Mentor Graphics Corporation, Siemens Industry Software Inc., or their affiliates (collectively, "Siemens"). The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the confidential and proprietary information.

This document is for information and instruction purposes. Siemens reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Siemens to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Siemens products are set forth in written agreements between Siemens and its customers. **End User License Agreement** — You can print a copy of the End User License Agreement from: mentor.com/eula.

No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Siemens whatsoever.

SIEMENS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

SIEMENS SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL OR PUNITIVE DAMAGES, LOST DATA OR PROFITS, EVEN IF SUCH DAMAGES WERE FORESEEABLE, ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF SIEMENS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

LICENSE RIGHTS APPLICABLE TO THE U.S. GOVERNMENT: This document explains the capabilities of commercial products that were developed exclusively at private expense. If the products are acquired directly or indirectly for use by the U.S. Government, then the parties agree that the products and this document are considered "Commercial Items" and "Commercial Computer Software" or "Computer Software Documentation," as defined in 48 C.F.R. §2.101 and 48 C.F.R. §252.227-7014(a)(1) and (a)(5), as applicable. Software and this document may only be used under the terms and conditions of the End User License Agreement referenced above as required by 48 C.F.R. §12.212 and 48 C.F.R. §227.7202. The U.S. Government will only have the rights set forth in the End User License Agreement, which supersedes any conflicting terms or conditions in any government order document, except for provisions which are contrary to applicable mandatory federal laws.

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' trademarks may be viewed at: www.plm.automation.siemens.com/global/en/legal/trademarks.html and mentor.com/trademarks.

The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

Support Center: support.sw.siemens.com

Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Revision History ISO-26262

Revision	Changes	Status/ Date
5.3	Modifications to improve the readability and comprehension of the content. Approved by Farshad Dailami. All technical enhancements, changes, and fixes listed in the Release Notes are reflected in this document. Approved by Tim Peeke.	Released October 2020
5.2	Modifications to improve the readability and comprehension of the content. Approved by Farshad Dailami. All technical enhancements, changes, and fixes listed in the Release Notes are reflected in this document. Approved by Tim Peeke.	Released July 2020
5.1	Modifications to improve the readability and comprehension of the content. Approved by Farshad Dailami. All technical enhancements, changes, and fixes listed in the Release Notes are reflected in this document. Approved by Tim Peeke.	Released April 2020
5.0	Modifications to improve the readability and comprehension of the content. Approved by Farshad Dailami. All technical enhancements, changes, and fixes listed in the Release Notes are reflected in this document. Approved by Tim Peeke.	Released January 2020

Author: In-house procedures and working practices require multiple authors for documents. All associated authors for each topic within this document are tracked within the Mentor Graphics Technical Publication's source. For specific topic authors, contact Mentor Graphics Technical Publication department.

Revision History: Released documents include a revision history of up to four revisions. For earlier revision history, refer to earlier releases of documentation on Support Center.

Table of Contents

Revision History ISO-26262

Chapter 1

Verification Management Overview	15
What is Not in this Manual	17
Verification Management for Analyzing Requirements	17
Modes of Use for Verification Management	19
XML Terms and Concept Review for Plan Import	19

Chapter 2

Capturing and Managing a Verification Plan	21
Introduction to the Verification Plan	21
Verification Plan Contents	23
Fields in the Verification Plan	23
Adding User Defined Data to Plan	26
Quick Overview to Linking with the Coverage Model	27
Overriding at_least Values in Testplan	28
Testplan Creation Tools for Productivity	28
Coverage Calculation in Testplans	29
Definition of Effective Weight and Actual Weight	29
Autoweight and the Determination of Effective Weight	30
Testplan Coverage Calculation Algorithm	32
Testplan Scope Coverage Calculation	32
Calculations for Empty Testplan Sections	33
Testplan vs. Design Hierarchy with Autoweight	33
Guidelines for Writing Verification Plans	35
Parameterizing Testplans	35
Guidelines for Excel Spreadsheets	36
Guidelines for Word Documents	37
Guidelines for DocBook	38
Recognized Paragraph Labels in Word and DocBook	39
Exporting a Plan to XML	41
Exporting From Microsoft Excel or Word	41
Exporting From DocBook	44
Importing an XML Verification Plan	45
Supported Plan Formats	45
Importing Plan from the Command Line	46
Importing Plan from the GUI	46
Importing a Hierarchical Plan	47
Adding Columns to an Imported Plan	48
Customized Import with xml2ucdb.ini Config File	49
Excluding Items from a Testplan	50

When Should Items be Excluded from a Testplan?	50
Excluding Linked Coverage with LinkWeight	50
Excluding a Testplan Section Using the Weight Column	51
Excluding Portions of a Plan	52
About Hierarchical Testplans	53
Mixing Autonumbered and Non-autonumbered Plans	53
Example of a Nested Plan	54
Inheritance in Hierarchical Plans	55
Merging Verification Plan with Test Data	56
Linking Verification Plan and Coverage Items	58
Linkable Design Constructs	58
Links Between the Plan Section and the Coverage Item	60
How Links are Established	60
Multiple Items and their Links	61
Datafield Rules and Requirements for Links	61
Specifying Levels of Hierarchy in Plan	62
What are Unimplemented Links and Why Use Them?	63
Counting the Coverage Contributions from Unimplemented Links	63
Weighting to Exclude Testplan Sections from Coverage	64
Backlink Coverage Items to Plan with SV Attributes	65
Items Available for Backlinking	65
Attribute Syntax for Backlinking	65
Module and Module Instance Syntax and Examples	66
Covergroup Instance Syntax and Example	67
Notes and Limitations on Backlinking	68
Finding Unlinked Items	68
Finding Information for Missing Verification Plan Links	69
Link Debugging Using coverage tag Command	70
Syntax for Links to the Coverage Model	72
Generic Link Entry Syntax Rules	72
All-Purpose coverage tag Link	73
Functional Coverage Links	74
Covergroups, Coverpoint and Crosses	74
Assertions, Directives, and Generic Coverage Items	76
Code Coverage Links	77
Bin Links	79
Identifying the Full Path to a Bin	79
Bin Coverage	80
Instances and Design Units Links	80
Directed Test Links through Test Records	80
Links for Classes	81
XML Links	83
Rules-based Linking for Tracking Verification Requirements	84
What is a Rule?	84
Rules Usage	84
Coverage Calculation of Rules	84
Rule Syntax	85
Storing and Displaying Rules	86
Currently Supported Rules	87

Table of Contents

TEST_ATTRIBUTE	88
GLOBAL_ATTRIBUTE	90
OBJECT_ATTRIBUTE	91
OBJECT_COVERED	92
COVERAGE	93
FORMAL_ATTRIBUTE	95
XML to UCDB Testplan Examples	97
Microsoft Word Example	97
Verification Management Windows	99

Chapter 3

Questa Testplan Creation Using OpenOffice/LibreOffice Calc Extension 101

Installing the Questa Calc Extension	101
Disabling the Questa Calc Extension	103
Re-Enabling a Disabled Questa Calc Extension	104
GUI Reference for the Questa Calc Extension	105
The Questa VM Menu	106
The Questa Toolbar	108
Settings Dialog	110
Using the Questa Calc Extension	112
Creating a New Testplan from Scratch	112
Generating a Testplan from a Testplan UCDB	115
Using an Existing Testplan as a Template	115
Managing Links	116
How the Links Work	116
Adding Links with the Select Link Dialog	116
Removing Existing Association with Testplan	119
Saving and Exporting to XML and UCDB	120
How and Why to Export TestPlan to XML	120
How and Why to Export Testplan to UCDB	120
Validation Checks Performed	121

Chapter 4

Questa Testplan Creation

Using Excel Add-In 123

Installing the Excel Add-In	123
Disabling the Questa Excel Add-In	125
Re-Enabling a Disabled Questa Add-In	126
GUI Reference for the Questa Add-In	127
The Questa VM Menu	128
The Questa Toolbar	130
Settings Dialog	132
Using the Questa Add-In	135
Creating a New Testplan from Scratch	136
Generating a Testplan from a Testplan UCDB	138
Using an Existing Excel Testplan as a Template	138
Managing Links	140
How the Links Work	140

Adding Links with the Select Link Dialog	140
Removing Existing Association with Testplan	143
Saving and Exporting to XML and UCDB	144
How and Why to Export TestPlan to XML	144
How and Why to Export Testplan to UCDB	144
Validation Checks Performed	145
Chapter 5	
Automated Run Management	147
Introduction to Run Management	147
Run Management Examples	148
Chapter 6	
Analyzing Test Results Based on a Plan	155
Preparing the UCDB File for Tracking	156
Test Data in the Tracker Window	157
Viewing Test Data in the Tracker Window	157
Invoking Coverage View Mode	158
Changing Goal, Weight and User Attribute Values in Tracker	159
Analyzing Results with coverage analyze	160
Refreshing Tracker after Changing a Plan	161
Storing User Attributes in UCDB	161
Filtering in the Tracker Window	163
Filtering Overview	163
Filtering Data in a Tracker Window	164
The Create Filter Dialog Box	166
Add/Modify Criteria Dialog Box	168
Retrieving Test Attribute Record Content	168
Analysis for Late-stage ECO Changes	169
Chapter 7	
Analyzing Verification Results	171
Use Models for Results Analysis	171
Use Flow	172
Data Analysis Tasks	173
Input Files for Results Analysis Database	174
Creating a Results Analysis Database (TDB)	175
Message Transformation	178
Transform Rules Files	178
Example or Template Rules Files	179
Editing Transform Rules Files in the GUI	179
Triage Command Examples	181
Triage Command Related Specifics	184
Lockfile	184
Tip for Grid Use to Avoid Non-deterministic Results	184
Automatic WLF and Logfile Import Search Order	184
About the .tdb Database for Results Analysis	186
The Testname Field and its Importance to Linking	186

Table of Contents

Organizing the Results Analysis Data	187
Setting Custom Hierarchy Configurations	187
Setting Custom Sorting Configurations	188
Setting Custom Analysis Question Queries	188
Filtering Data in Results Analysis Database	189
Filtering Data in Results Analysis or Message Viewer Windows	190
Edit Filter Expression Dialog Box	190
Viewing Results Analysis Data in Text Reports	192
Creation of the Triage Report	192
Filtering the Text Report Output	192
Counts and Compression in the Triage Report	192
Triage Report Formats	193
Viewing vcover diff Results in the Results Analysis Window	194
Predefined Fields in the WLF and UCDB	195
Overview of Predefined Fields	195
Message Table Predefined Fields	196
Test Table Predefined Fields	197
User-defined Fields	198
Transform Rules File Format	199
Syntax of transform and field Constructs	199
Transform Examples	200
Matching Messages Spanning Multiple Lines	201
Debugging the Transforms	203
Levels of Debugging	203
LOG File Transforms	204
The Transform Buffer and Buffer Management	204
Managing Multi-line Messages	205
Buffer Flow	205
 Chapter 8	
Analyzing Trends	207
Overview and Use Flow of Trend Analysis	208
What is a Trend UCDB?	208
Use Flow	208
Objects Available for Trend Analysis	209
Coverage Computation for Trend Analysis	209
Creating a Trend UCDB	210
Use Scenario for Adding Attributes to a Trend UCDB	211
Creating a Trend Report	212
Viewing Trend Report Data in the Trender Window	213
Trending Usage FAQs	214
 Appendix A	
xml2ucdb.ini Configuration File	217
Overview	218
When to Modify the xml2ucdb.ini File?	218
XML Version Support	218
xml2ucdb.ini File Format	218

varfile Setting for Parameterizing Testplans	222
Parameters Customizing the Testplan Import	223
Parameters Controlling the Configuration File.	223
Parameters Controlling the Inclusion of Data.	224
Parameters Identifying Section or Column Boundaries	225
Parameters Mapping Testplan Data Items	226
Parameters for Associating with Specific Tags	226
Parameters Mapping by Attribute Name	227
Parameter for Mapping by Column Sequence	227
Parameter for Mapping by Explicit Label	230
Case Sensitivity and Field Mapping and Attribute Names.	231
Parameters Determining Hierarchy and Section Numbering	232
Parameters Adding Tag-based Prefixes to Section Numbers	233
Parameters Specifying UCDB Details	234
Parameters Specifying a Pre-Process XSL Transformation	234

End-User License Agreement
with EDA Software Supplemental Terms

List of Figures

Figure 1-1. Verification of a Design	16
Figure 1-2. Verification Management Flow	17
Figure 1-3. Verification Management Test Tracker	18
Figure 2-1. Testplan with Parameters	35
Figure 2-2. Excel Spreadsheet Example Verification Plan	42
Figure 2-3. XML Example of XML Output	43
Figure 2-4. Verification Plan Linking	60
Figure 2-5. Multi-Link and Multi-Type Example	61
Figure 2-6. Multi-Link, Single-Type Example	61
Figure 2-7. Finding Link Information for Verification Plan	70
Figure 2-8. Partial Source Document for Word.	98
Figure 3-1. Questa Toolbar in OpenOffice/LibreOffice Calc	103
Figure 3-2. OpenOffice/LibreOffice Extension Manager Dialog	103
Figure 3-3. Questa VM Menu for OpenOffice/LibreOffice Calc	106
Figure 3-4. Questa Toolbar	108
Figure 3-5. Questa Addon Settings Dialog Box	110
Figure 3-6. Enter Testplan Details Dialog.	113
Figure 3-7. Select Link Dialog	117
Figure 3-8. Adding a Rule with Select Link	118
Figure 3-9. Add Rule Dialog Box	118
Figure 3-10. Adding a Sub-testplan with Select Dialog Box	119
Figure 3-11. Files of type: UCDB File Version.	121
Figure 4-1. Questa Tab in Excel 2007/2010/2013.	124
Figure 4-2. Office button and Excel Options.	125
Figure 4-3. Excel Options menu	125
Figure 4-4. Excel Manage dropdown menu.	126
Figure 4-5. Questa VM Menu for Excel Add-In	128
Figure 4-6. Questa Toolbar: Features and Icons	130
Figure 4-7. Settings Dialog Box	132
Figure 4-8. Select Links Dialog.	141
Figure 4-9. Selecting Bins for Linking	141
Figure 4-10. Adding a Rule with Select Link	142
Figure 4-11. Add Rule Dialog Box	142
Figure 4-12. Adding a Sub-testplan with Select Dialog Box	143
Figure 5-1. RMDB File Example	148
Figure 5-2. fpu_conf.rmdb.	149
Figure 6-1. Test Data in Verification Tracker Window.	158
Figure 6-2. Filtering Displayed Data	166
Figure 7-1. Overview of Results Analysis Flow	172
Figure 7-2. Verification Results Analysis Window.	174

Figure 7-3. Create Results Analysis Database Dialog Box	176
Figure 7-4. Transform Rule File Editor	180
Figure 7-5. Editing New Transform Rule from a Message	181
Figure 7-6. Configuring the Message Data Hierarchy	187
Figure 7-7. Display of Results by Category, Severity, then Message	188
Figure 7-8. Creating a Sort Configuration	188
Figure 7-9. Creating an Analysis Question	189
Figure 7-10. Edit Filter Expression Dialog Box	191
Figure 7-11. Example Triage Report	192
Figure 7-12. vcover diff Results in the Verification Results Analysis Window	194
Figure 8-1. Create Trend Database Dialog Box.	211
Figure 8-2. Trender Window	213
Figure 8-3. Trend Graph	214
Figure A-1. Default xml2ucdb.ini Sample.	220
Figure A-2. Adding User Defined datafields to the xml2ucdb.ini Configuration File	229

List of Tables

Table 2-1. Default (expected) Columns/Fields for an Imported Plan	24
Table 2-2. Recognized Paragraph Labels	39
Table 2-3. Plan Weight and LinkWeight Usage for Excluding Coverage	50
Table 2-4. parent.xml	54
Table 2-5. child.xml	54
Table 2-6. Nested Testplan	55
Table 2-7. Recognized (Linkable) Coverage Design Constructs	58
Table 2-8. Sample Testplan Section for a Generic Tag Link	73
Table 2-9. Accepted Formats for covergroup, coverpoint, and cross Entries	74
Table 2-10. Assertion and CoverDirective Links	76
Table 2-11. Code Coverage Link Details	77
Table 2-12. Class Example Linking	82
Table 3-1. Questa VM Popup Menu	106
Table 4-1. Questa VM Popup Menu	128
Table 6-1. Content Description of the Add/Modify Criteria Dialog Box	168
Table 7-1. Transform Rules File Editor Toolbar Buttons	181
Table 7-2. Predefined Fields from Message Table	196
Table 7-3. Predefined Fields from Test Table	197
Table A-1. Location and Extraction Meta-parameters	223
Table A-2. Parameters for ID of Item Containing Testplan	224
Table A-3. Parameters for ID of Testplan Section and Column Boundaries	225
Table A-4. Parameters for Mapping by Tag Association	226
Table A-5. Data Value Extracted from Named Attribute	227
Table A-6. Predefined Column Label (Reserved) -datafield Keywords	228
Table A-7. Parameters for Hierarchy and Numbering	232
Table A-8. Parameters for Designating a Stylesheet: -sectionprefix	233
Table A-9. Parameters for Hierarchy and Numbering - UCDB Details	234
Table A-10. Parameters for Designating a Stylesheet	234

Chapter 1

Verification Management Overview

Verification Management is a set of functionality within Questa SIM which offers a wide variety of features for managing your test environment.

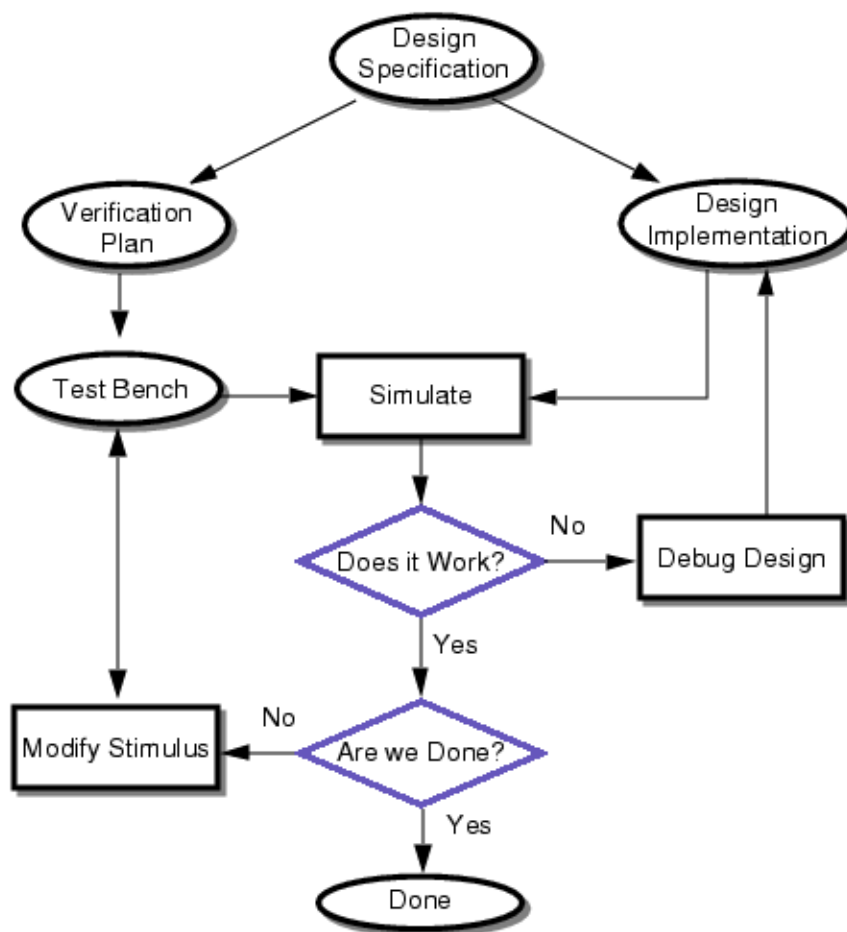
The Verification Management features are available in the GUI with the Verification Management Browser and Tracker windows and are built upon a database called the Unified Coverage DataBase (UCDB). The UCDB database has been developed to store all the information necessary to manage the complete verification flow. The UCDB is used natively within Questa SIM to save all code coverage, functional coverage and assertion data from simulation. Refer to [“What is the Unified Coverage Database?”](#) in the User’s Manual for information on UCDBs.

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:

- 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	17
Verification Management for Analyzing Requirements	17
Modes of Use for Verification Management	19

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.

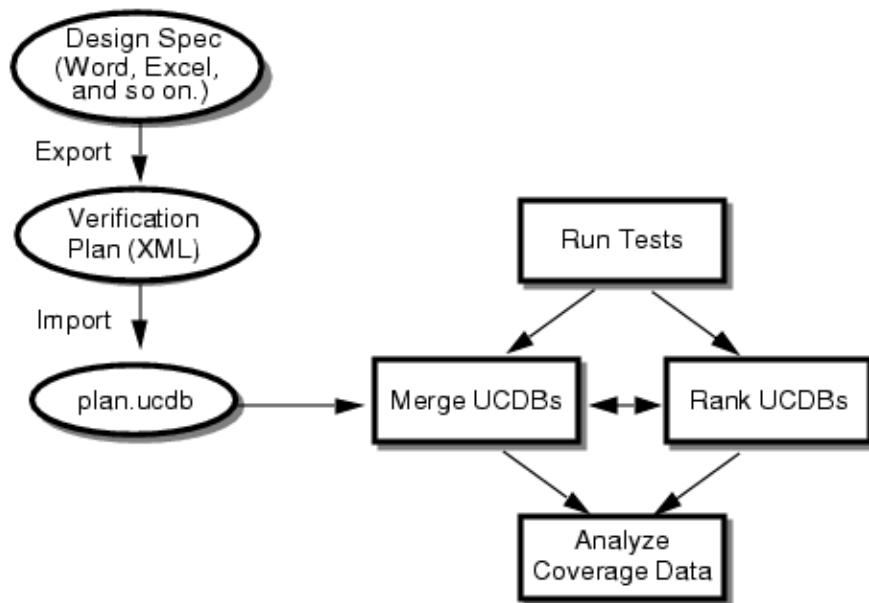
If this is not the case, please see the *Questa SIM User's Manual* for important information relating to the generation of UCDBs, merging, ranking, and the Verification Management Browser.

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 of this process is illustrated in [Figure 1-2](#).

Figure 1-2. Verification Management Flow



The flow is as follows:

1. Create the verification plan, export it to .xml format, and convert to a UCDB file. Refer to [“Exporting a Plan to XML”](#) and [“Importing an XML Verification Plan.”](#)

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

2. Run tests, capturing coverage from each in a UCDB file. Refer to “[Collecting and Saving Coverage Data](#)” and “[Rerunning Tests and Executing Commands](#)” in the User’s Manual.
3. Merge the verification plan UCDB file and the coverage UCDB files from each run. Refer to “[Merging Coverage Data](#)” in the User’s Manual.

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.

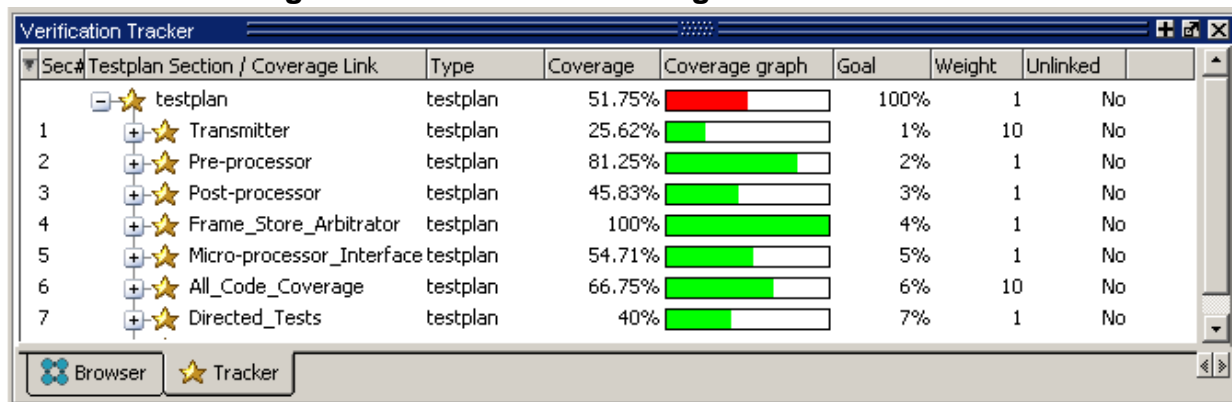
4. Rank tests, either before or after merging with the verification plan into a single .ucdb.
5. Load the UCDB file in Coverage View mode, using [vsim -viewcov](#).

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 (refer to “[Coverage View Mode and the UCDB](#)” in the User’s Manual). This is why coverage analysis is not available in vsim (simulation) or vcover (the standalone batch utility.) In vsim simulation, UCDB files are created only 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 with the "[coverage analyze](#)" command.

Refer to [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



Refer to “[Calculation of Total Coverage](#)” in the User’s Manual for details on how coverage numbers are calculated wherever Total Coverage figures are displayed. The referenced section

includes subsections detailing the calculation of numbers in the Verification Tracker and the Verification Browser.

Modes of Use for Verification Management

You can use one of several modes when accessing the Verification Management tools, depending on the level of interaction you desire with the UCDB that stores the verification data. The UCDB database itself is stored in a persistent form in a binary file.

You access the data within the UCDB from one of three basic modes when using Questa SIM utilities:

- 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 “[Verification Management Flow](#)” 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 is 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, denoting that the string is a title. 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 key component in any verification management endeavor is a verification plan, also known as a testplan. This section describes the content and structure, as well as the syntax of the elements of that plan.


Introduction to the Verification Plan.....	21
Verification Plan Contents	23
Testplan Creation Tools for Productivity	28
Coverage Calculation in Testplans.....	29
Guidelines for Writing Verification Plans.....	35
Exporting a Plan to XML	41
Importing an XML Verification Plan	45
Adding Columns to an Imported Plan.....	48
Customized Import with xml2ucdb.ini Config File	49
Excluding Items from a Testplan	50
About Hierarchical Testplans.....	53
Merging Verification Plan with Test Data	56
Linking Verification Plan and Coverage Items	58
Syntax for Links to the Coverage Model.....	72
Rules-based Linking for Tracking Verification Requirements.....	84
XML to UCDB Testplan Examples	97
Verification Management Windows.....	99

Introduction to the Verification Plan

At the heart of verification management is the verification plan, written in an editing tool and converted into a UCDB file.

Generally, you write a verification plan using standard editing tools: Microsoft Word or Excel, Adobe Framemaker, or requirements tracking software. After it is written, the plan is then converted to a UCDB, enabling you to take advantage of the many analysis features inherent to Verification Management.

Tip

 Questa SIM offers an easy to use tool to aid you in creating testplans, linking to coverage objects, and annotating coverage data. It quickly and easily performs automatic conversions for Excel and OpenOffice Calc formatted plans. See “[Questa Testplan Creation Using Excel Add-In](#)” for more information.

The architecture of the UCDB allows it to store sections of a Verification Plan as a tree hierarchy of scopes, similar to how it stores a design hierarchy of 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. Additionally, you can interface with this infrastructure via the UCDB API and develop very powerful tracking capacities according to your particular requirements.

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. 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.

One of the most important aspects of the Verification plan is the definition of links 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.

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 that follow focus on using Microsoft Word and Excel to capture this information and import it into a UCDB file.

Fields in the Verification Plan	23
Adding User Defined Data to Plan	26
Quick Overview to Linking with the Coverage Model.....	27
Overriding at_least Values in Testplan	28

Fields in the Verification Plan

The fields (or columns, in the case of a spreadsheet) of data in a testplan are determined positionally. In other words, for an import to work correctly, the fields to be imported must appear in their default (expected) positions in the plan.

These positions are set, and can be edited, with the “datafields” parameter inside the *xml2ucdb.ini* file (the configuration file governing the import process), or the -datafields parameter to the [xml2ucdb](#) command.

The expected data fields are those which appear without any editing of the *xml2ucdb.ini* file. Those fields are listed in order of their expected position within the testplan in [Table 2-1](#). The ATLeast, LinkWeight, Unimplemented, LinkExclusion, and LinkExclusionComment data fields (shaded in table below) are, by default, commented out in the *xml2ucdb.ini* file. You must add these fields in order for them to appear in the UCDB.

For details related to editing the fields and their expected order, see [Parameters Mapping Testplan Data Items](#).

Every entry (string) in any field/column of the plan is delimited by whitespace (space, tab, or return) or a semi-colon (“;”).

Table 2-1. Default (expected) Columns/Fields for an Imported Plan

Data Fields / Columns	Format	Description
Section	<section#>[.<subsection#>] ...	The section number, which is used to determine plan hierarchy and to generate tags (links to coverage items in the design). Duplicate Section numbers are not allowed and result in an error. You must define parent sections before child sections, for example: section 1 before 1.1, before 1.1.1, and so on.
Title	<scope_name>	The name of the testplan “section”, where <scope_name> appears in the VM Tracker window and in the coverage reports. Use of special characters is discouraged (“*?/.”). See “Quick Overview to Linking with the Coverage Model” . Duplicate Title entries (at a sibling level) are not allowed and result in an error. An example of a title/name: 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 / <i>Interfaces/Wishbone</i> .
Description	<text>	Textual description of the 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 the coverage objects or test data records which are linked to the testplan section. A testplan section can have one or multiple links to a coverage object or test data record. A specific format is required for the Link string itself. See “Syntax for Links to the Coverage Model” for the format for each type of link. See also “Specifying Levels of Hierarchy in Plan” .

Table 2-1. Default (expected) Columns/Fields for an Imported Plan (cont.)

Data Fields / Columns	Format	Description
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). 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.
Weight	<weight_as_int>	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. 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, an absolute path be specified in the Link column (for example, /top/a/cov1).
AtLeast	<AtLeast_value>	Overrides the value of 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” .
LinkWeight	<weight>	Specifies weight for a specific link; overrides the general weight for that coverage object.
Unimplemented	Yes No <integer> Yes — one bin created No — none (default) <integer> — # of cover items to be created	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?” .

Table 2-1. Default (expected) Columns/Fields for an Imported Plan (cont.)

Data Fields / Columns	Format	Description
LinkExclusion	<coverage_item_string> <coverage_item_string> ...	To exclude a child scope or bin of a linked coverage item: this column contains the full or relative path with respect to the linked coverage item. Entries are delimited either by whitespace (space, tab, or return) or a semi-colon (";"). See also " Excluding Items from a Testplan ".
LinkExclusionComment	<exclusion_comment>	Specifies a comment for the excluded child scope or bin item defined in LinkExclusion.

A specific format is required for some types of coverage (covergroups, coverpoints, and so on). See [Syntax for Links to the Coverage Model](#) for details of each type.

The information in the Link, Type, and Path columns is used directly for tagging the defined coverage object. Using this data, a command line is automatically calculated for the "coverage tag" command by the xml2ucdb utility. This command is then stored as an attribute of the testplan section, and used later during the merge process to automatically tag the coverage objects within the simulation UCDBs. Once a testplan section has linked the 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.

Naming Conventions for Testplan Sections

The names of testplan sections (entered in the "Title" column of the testplan) are used to generate a document hierarchical path name, much in the same way as a design has a hierarchical name. It is recommended that you not use special characters within section names: such as "?" or "*" (they can be confused as wildcards) and "." or "/" (they can be confused as delimiters). Use of these characters can lead to problems in the CLI and within the tracker window.

Adding User Defined Data to Plan

The fields /columns of data in the Verification Plan represent the pre-defined and expected standard attributes within the UCDB. In addition to these attributes, you can add your own data to include for later analysis. For example, you might want to add the name of the person responsible for a particular section of the plan, or the priority level for a section of the testplan.

You can include user-defined data directly into the source plan, before you import it to a UCDB.

Procedure

- Adding data to source plan, before import:
 - a. Add the desired field data to the verification plan (Excel, Word, and so on).
 - b. Edit the settings for the expected datafields.

When you want to include a field or column of data in your verification plan, you must also instruct the `xml2ucdb` where these fields are expected to be in the source plan, and thus where they should be placed in the resulting UCDB. Do this using the `-datafields` argument to the command, or the “datafields” parameter within the `xml2ucdb.ini` configuration file. See [Parameters Mapping Testplan Data Items](#) for details.

- c. Import your plan.
- Alternatively, you can add data to the UCDB after it has been imported by storing the user attribute:

coverage attribute -test <testname> -name <attr_name> -value <attr_value>

Results

The values of the attributes you added to the plan become available for use within the Questa SIM Tracker window or the command line querying to filter the results, based upon your own unique data.

Quick Overview to Linking with the Coverage Model

One of the most important aspects of the Verification Plan is the definition of the links 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 UCDBcolumn.
- 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.

Testplan Creation Tools for Productivity

Two programs are available which greatly assist in the creation and editing of testplans. One is an OpenOffice Extension for Testplans, or an Excel Add-In for Testplans.

Using either of these programs is, by far, the easiest method for creating a testplan — either from scratch, or taking an existing testplan and using it as a template for creating a new testplan. These plug-in programs are distributed along with the Questa SIM installation, located within the product install directory.

OpenOffice Extension for Testplans

For complete instructions on the installation and use of the OpenOffice extension see “[Questa Testplan Creation Using OpenOffice/LibreOffice Calc Extension](#)”.

Excel Add-In for Testplans

To install the Add-In, run the following:


<install_dir>/vm_src/QuestaExcelAddIn.msi

For basic installation and complete usage information, see “[Questa Testplan Creation Using Excel Add-In](#)”. Further detailed instructions on installing the program are available in the file *<install_dir>/vm_src/HowToInstallExcelAddIn.txt*.

Coverage Calculation in Testplans

Coverage calculations algorithms for testplans and testplan sections, as well as the topics relevant to them are described in this section.

Note

 The coverage calculation algorithms applied to testplans and testplan sections are not the same as those applied to “Total Coverage” calculations, or those applied to calculations of instances and/or design units. Refer to “[Calculation of Total Coverage](#)” in the User’s Manual for details of these aggregation algorithms.

Definition of Effective Weight and Actual Weight	29
Autoweight and the Determination of Effective Weight	30
Testplan Coverage Calculation Algorithm	32
Testplan vs. Design Hierarchy with Autoweight.....	33

Definition of Effective Weight and Actual Weight

The *effective weight* of a testplan item is the weight of a testplan scope or a linked coverage item that is used in the coverage calculation of a testplan. By default the effective weight of an item is the same as the actual weight of that item.

The *actual weight* of a testplan item is defined by the language (such as covergroups, coverpoints, and crosses for the SystemVerilog language), implicitly assigned by the tool, or explicitly set by the user in a coverage object by using the coverage weight command.

For example, the actual weight of a covergroup is specified in the covergroup declaration. If nothing is specified, then the SystemVerilog LRM defined default value is used. Some coverage objects — like module instances, toggles, FSMs, and so on — get their actual weights from the tool’s default value of 1. The user can modify any actual weight by using the coverage weight CLI command.

Actual weights are used for the coverage calculation of design hierarchy and effective weights are used for the coverage calculation of testplan hierarchy.

Related Topics

[Coverage Calculation in Testplans](#)

[Autoweight and the Determination of Effective Weight](#)

[Fields in the Verification Plan](#)

Autoweight and the Determination of Effective Weight

The effective weight of a testplan scope or item depends on whether you are using the default weighting method or the autoweighting method (which you enable by using the `xml2ucdb - autoweight` argument).

Default Testplan Coverage

In the default testplan coverage calculation, the effective weight of:

- a testplan scope — is specified in the `Weight` column for that testplan scope. If nothing is specified then the effective weight is 1;
- a linked coverage item — is the actual weight of that linked coverage item when nothing is specified in the `LinkWeight` column. If any weight is specified in the `LinkWeight` column for a linked coverage item then the effective weight is the value specified in that `LinkWeight` column. So the value specified in the `LinkWeight` column for a linked coverage item has higher priority than the actual weight of the linked coverage item.

Autoweight Testplan Coverage

In autoweight testplan coverage calculation, the effective weight of:

- a testplan scope — is equal to the total number of bins present in all the locally linked coverage items and all recursive child testplan scopes.

If a testplan item is empty then the effective weight of that scope is 1. That is done to negatively contribute coverage to the parent scope, as unlinked testplans are treated as error cases.

Autoweighting ignores explicit weights specified in the `Weight` column for a testplan scope, with the exception of zero weights. The zero weight is the primary mechanism for excluding a testplan scope, hence only that value has a higher priority than the number of bins for calculating the effective weight of a testplan.

- a linked coverage item — is equal to the number of bins present in that linked coverage item. If the linked coverage item is a directed test then the effective weight of that item is 1, considering there is a single bin associated with that linked item.

If either a 0 weight is specified in the `LinkWeight` column or the actual weight of a linked item is 0, then the effective weight will be 0. The only way the children can propagate up 0 weights is if they are all excluded. The algorithm automatically excludes a parent scope if all its children are excluded and there are no locally linked coverage items.

Related Topics

[Coverage Calculation in Testplans](#)

Definition of Effective Weight and Actual Weight

Testplan Coverage Calculation Algorithm

The weight of each testplan section is used to compute the coverage of the parent testplan section according to the “weighted average” method — regardless of whether that section’s weight is determined by the default method or autoweighting. 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

Without autoweighting, using the weighted average calculation, the coverage calculation for the parent testplan section is 50%.

In other words, the autoweighting and the default method of testplan coverage calculation both use the same weighted average method of coverage calculation. However, the effective weights in the autoweighting case are determined by the number of bins present in the testplan scope's associated coverage objects. The difference between the two methods is in the determination of the effective weights, not in the coverage calculation formula.

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.

Testplan Scope Coverage Calculation	32
Calculations for Empty Testplan Sections.....	33

Testplan Scope Coverage Calculation

The coverage of a testplan scope is calculated by taking the weighted average of all child testplan scopes using the effective weight of each item. If a testplan scope contains child testplans with coverage links, then the cumulative coverage of all the linked items is calculated first by taking the weighted average using the effective weights of those linked items. That cumulative coverage is then used with either a weight of 1 (default) or a weight equal to the total number of bins of those linked items (using autoweight), along with the coverage of child testplans to determine the weighted average for calculating the coverage of the parent testplan.

Calculations for Empty Testplan Sections

Empty testplan sections trigger an additional algorithm for coverage calculation. The following calculations apply:

- If a testplan section (scope) is empty due to a missing coverage link, the testplan will have 0% coverage and that coverage is contributed to the parent testplan, if the testplan itself has a non-zero weight.
- If none of a testplan section's linked items have non-zero weights, the following occurs:
 - The testplan section does not contribute to the coverage computation (of the parent testplan section)
 - If the testplan section's effective weight is non-zero, a value of 0% is displayed by the GUI and coverage reports
 - If the testplan section's effective weight is zero, a value of 100% is displayed by the GUI and coverage reports

Note



For any of the above mentioned cases, the testplan is greyed-out in the Tracker window.

Testplan vs. Design Hierarchy with Autoweight

A UCDB file may contain two different type of hierarchies:

- Design hierarchy - for the design itself, which is the source of coverage data.
- Testplan hierarchy - for different testplan scopes created from testplan specifications for tracking purposes. There are unidirectional links from the testplan hierarchy to the design hierarchy which creates the links between coverage objects and testplan scopes.

The coverage of an object in the design hierarchy is calculated by using a single algorithm. The algorithm calculates the weighted average of different coverage types present in that object. In this case the coverage numbers of each coverage type (like statement, branch, expression, condition, covergroup, and so on) are calculated first, and then the weighted average is applied on those coverage numbers by using the weights for different coverage types. By default the weights of all coverage types is 1. The user can set those weights away from 1 by using the coverage weight CLI command. So the coverage of a design hierarchy object (like module instance, fsm, covergroup, and so on) is the same irrespective of whether that is calculated for the design hierarchy itself or for the testplan hierarchy (when that object is linked to a testplan scope). It is not possible to have different coverage numbers of a design hierarchy object in different places (unless a bug is present).

The coverage of the testplan hierarchy is calculated by starting with the coverage of linked coverage objects which are part of the design hierarchy. Those coverage number are calculated

by a fixed algorithm as mentioned above, and then those coverage numbers roll up in the testplan hierarchy. The process of that rolling up in the testplan hierarchy uses either the default algorithm or the autoweighting algorithm depending on whether xml2ucdb is invoked with -autoweight option or not. Hence, the autoweighting method has its effect on coverage calculation of testplan hierarchy, it does not have any effect on the coverage calculation of the design hierarchy.

Guidelines for Writing Verification Plans

This section contains a set of guidelines for writing verification plans to ensure that the Verification Plan Import utility properly recognizes the data in your plan.

For examples of plans written in Excel, Word and DocBook, and converted from XML to UCDB, see “[XML to UCDB Testplan Examples](#)”.

Parameterizing Testplans 35

Guidelines for Excel Spreadsheets 36

Guidelines for Word Documents 37

Guidelines for DocBook..... 38

Recognized Paragraph Labels in Word and DocBook..... 39

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 (such as Excel or Word) 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

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 following 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

[Testplan Creation Tools for Productivity](#)

[varfile Setting for Parameterizing Testplans](#)

[Parameters Mapping Testplan Data Items](#)

[Importing an XML Verification Plan](#)


Guidelines for Excel Spreadsheets

Because the columns of data in an Excel-based imported testplan are determined positionally, all columns of data must appear in their default (expected) positions in the plan.

The expected data fields, listed in the order of their expected position within the testplan, are shown in [Table 2-1](#).

The positions are set in the “datafields” parameter inside the *xml2ucdb.ini* file, or using the -datafields option with the xml2ucdb command.

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).

- Each row in Excel is interpreted as a testplan section, starting with the first¹ row.
- Excel testplans are required to have a “TYPE” entry corresponding to each “LINK” entry.
- Any rows whose first data field contains only a pound sign (#) is interpreted as a comment row, and the contents of that row are not included in the imported UCDB.
- 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.

Related Topics

[Parameters Mapping Testplan Data Items](#)

[Exporting a Plan to XML](#)

[Importing an XML Verification Plan](#)

Guidelines for Word Documents

Microsoft Word supports hierarchical sections using the “Heading” styles.

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

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”.

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

 Important: 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 "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.

Exporting a Plan to XML

The information in this section is provided as an alternative method for customizing testplan import.

The easiest method for creating/editing verification plans is to use the “[Excel Add-In for Testplans](#)”. Please refer to that section for instructions and usage information.

Before you can import a verification plan, if not using the Questa Excel Add-In 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.

Exporting From Microsoft Excel or Word	41
Exporting From DocBook.....	44

Exporting From Microsoft Excel or Word

You can use Microsoft Excel, Microsoft Word, or DocBook to export a plan.


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](#)” or “[Guidelines for Word Documents](#)”.

Procedure

1. Open the spreadsheet or document.
2. Save as an .xml file:
 - Choose **File > Save As**. This displays the Save As dialog box.
 - In the **Save as type** pulldown menu, choose **XML <spreadsheet/doc> 2003 (*.xml)**. This automatically changes the suffix of the spreadsheet or document to .xml.

Tip

 If you are using Excel 2007 or 2010, you must select **XML Spreadsheet 2003** as the Save as type pulldown option. Note that this is different than the **XML Data** save option.

- Click **Save**.

This saves the output as an *.xml* file.

If you receive a warning message because the file “may contain features that are not compatible with XML Spreadsheet 2003” click **Yes**, which ignores the warning.

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

Section	Title	Description	Link	Type	Weight	Goal
1	Transmitter	The transmitter is able to transmit ...			10	100
1.1	Bonding_MODE_0	This mode provides initial parameter ...	cover_fsm_idle_to_neg	Directive	1	100
1.2	Bonding_MODE_1	This mode supports user data rates ...	cover_fsm_idle_to_build	Directive	1	100
1.3	Bonding_MODE_2	This mode supports multiples of 63/64 ...	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 ...	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	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 the transmitter shall send 15 bit to the ...	monitor_channel_data:CRC;	CoverPoint	1	100

Excel XML Output

Once you export the Excel spreadsheet verification plan into XML, the output appears as shown in Figure 2-3 (some code has been greyed out as it is not useful to this example):

Figure 2-3. XML Example of XML Output

```

<Worksheet ss:Name="Sheet1">
  <Table ss:ExpandedColumnCount="8" ss:ExpandedRowCount="73" x:FullColumns="1"
    x:FullRows="1">
    <Column ss:AutoFitWidth="0" ss:Width="24.75"/>
    <Column ss:AutoFitWidth="0" ss:Width="117.75" ss:Span="1"/>
    <Column ss:Index="4" ss:AutoFitWidth="0" ss:Width="306"/>
    <Column ss:AutoFitWidth="0" ss:Width="216"/>
    <Column ss:AutoFitWidth="0" ss:Width="97.5"/>
    <Column ss:AutoFitWidth="0" ss:Width="72.75" ss:Span="1"/>
    <Row ss:AutoFitHeight="0" ss:Height="20.25">
      <Cell ss:Index="2" ss:StyleID="s21"><Data ss:Type="String">Verification Plan For G
      <Cell ss:StyleID="s21"/>
      <Cell ss:Index="5" ss:StyleID="s22"/>
    </Row>
    <Row ss:AutoFitHeight="0" ss:Height="13.5">
      <Cell ss:Index="5" ss:StyleID="s22"/>
    </Row>
    <Row ss:AutoFitHeight="0" ss:Height="18.75">
      <Cell ss:Index="2" ss:StyleID="s23"><Data ss:Type="String">Section</Data></Cell>
      <Cell ss:StyleID="s24"><Data ss:Type="String">Title</Data></Cell>
      <Cell ss:StyleID="s23"><Data ss:Type="String">Description</Data></Cell>
      <Cell ss:StyleID="s25"><Data ss:Type="String">Link</Data></Cell>
      <Cell ss:StyleID="s23"><Data ss:Type="String">Type</Data></Cell>
      <Cell ss:StyleID="s24"><Data ss:Type="String">Weight</Data></Cell>
      <Cell ss:StyleID="s23"><Data ss:Type="String">Goal</Data></Cell>
    </Row>
    <Row ss:AutoFitHeight="0" ss:Height="26.25">
      <Cell ss:Index="2" ss:StyleID="s26"><Data ss:Type="String">1</Data></Cell>
      <Cell ss:StyleID="s27"><Data ss:Type="String">Transmitter</Data></Cell>
      <Cell ss:StyleID="s28"><Data ss:Type="String">The transmitter is able to transmit
      <Cell ss:StyleID="s29"/>
      <Cell ss:StyleID="s28"/>
      <Cell ss:StyleID="s30"><Data ss:Type="Number">10</Data></Cell>
      <Cell ss:StyleID="s28"><Data ss:Type="Number">100</Data></Cell>
    </Row>
    <Row ss:AutoFitHeight="0" ss:Height="77.25">
      <Cell ss:Index="2" ss:StyleID="s26"><Data ss:Type="String">1.1</Data></Cell>
      <Cell ss:StyleID="s29"><Data ss:Type="String">Bonding_MODE_0</Data></Cell>
      <Cell ss:StyleID="s28"><Data ss:Type="String">This mode provides initial paramete
      <Cell ss:StyleID="s29"><Data ss:Type="String">cover_fsm_idle_to_neg</Data></Cell>
      <Cell ss:StyleID="s28"><Data ss:Type="String">Directive</Data></Cell>
      <Cell ss:StyleID="s30"><Data ss:Type="Number">1</Data></Cell>
      <Cell ss:StyleID="s28"><Data ss:Type="Number">100</Data></Cell>
    </Row>
    <Row ss:AutoFitHeight="0" ss:Height="102.75">
      <Cell ss:Index="2" ss:StyleID="s26"><Data ss:Type="String">1.2</Data></Cell>
      <Cell ss:StyleID="s29"><Data ss:Type="String">Bonding_MODE_1</Data></Cell>
      <Cell ss:StyleID="s28"><Data ss:Type="String">This mode supports user data rates
      <Cell ss:StyleID="s29"><Data ss:Type="String">cover_fsm_idle_to_build</Data></Cell>
      <Cell ss:StyleID="s28"><Data ss:Type="String">Directive</Data></Cell>
      <Cell ss:StyleID="s30"><Data ss:Type="Number">1</Data></Cell>
      <Cell ss:StyleID="s28"><Data ss:Type="Number">100</Data></Cell>
    </Row>
  </Table>
</Worksheet>
  
```

Worksheet Title

Document Title

Column Headings

First Row of Data

Second Row of Data

- Worksheet Title — based on the title of the Excel sheet.
- Document Title — based on the first row of the Excel spreadsheet.

- Column Headings — based on the third row of the Excel spreadsheet, identifying the headings of your plan.
- First Row of Data — based on the fourth row of the Excel spreadsheet, identifying the cell data associated with the Transmitter section of your plan.
- Second Row of Data — based on the fifth row of the Excel spreadsheet, identifying the cell data associated with the Bonding_MODE_0 section of your plan.

Related Topics

[Guidelines for Writing Verification Plans](#)

[Importing an XML Verification Plan](#)

[Merging Verification Plan with Test Data](#)

Exporting From DocBook

Because DocBook uses XML as its native 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	45
Importing Plan from the Command Line	46
Importing Plan from the GUI	46
Importing a Hierarchical Plan	47

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 you want to import data from one specific Excel spreadsheet, use the `-excelsheet <sheet_name>` argument. The `<sheet_name>` is the exact string as it appears in the tab (“Sheet1”, “Sheet2”, and so on) at the bottom of the Excel spreadsheet.

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


You can import a verification plan from the GUI:

1. Choose **View > Verification Management > Browser**.
2. Right-click anywhere inside the Verification Browser window and choose **Testplan 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.

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.

Adding Columns to an Imported Plan

Each attribute corresponds to a column with the same name in your verification plan.

Questa SIM requires that each plan being imported contain the pre-defined attributes defined in [Table 2-1](#).

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. Refer to [“Filtering Results by User Attributes”](#) in the User’s Manual for instructions.

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

- Use `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.
- Copy `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 as follows:

- Use one of the tools listed in “[Supported Plan Formats](#)”.
- Follow 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). Refer to “[xml2ucdb.ini Configuration File](#)” for more information on this file.

Excluding Items from a Testplan

You may want the coverage from certain items in a testplan to be excluded from counting toward the calculations. Several methods exist for excluding information that exists in a testplan from the UCDB, depending on what you want to exclude.

When Should Items be Excluded from a Testplan?	50
Excluding Linked Coverage with LinkWeight	50
Excluding a Testplan Section Using the Weight Column	51
Excluding Portions of a Plan.	52

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 (cont.)

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 LinkWeight, you must also:

- o 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:

- o enter a single 0 value in the LinkWeight column — which applies to all items listed in that row (see row Section4) so that any coverage items in that section are excluded from the testplan coverage score; or
- o enter the same number of 0 values as there are coverage items you want to exclude in that row within the LinkWeight column. In the table above, a 0 is entered for the cov3 link in Section 2 and for the cov6 link in Section 3. 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 section, set the value of the Weight column for that section to 0 within the overall testplan.

See “[Coverage Calculation in Testplans](#)” for details on weighting and calculations.

Excluding Portions of a Plan

It is possible to exclude portions of the source XML of the plan from being processed by using the "excludetags" parameter. This is a useful feature when you have control over the XML file format itself; for example, you can add or identify XML tag names within the source XML file that you want xml2ucdb to skip processing. This is more likely to be useful in a word processing type document where the application allows different formats to be added and they can then be identified in the XML source document at XML tags.

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:
 - o adding the "excludetags <tags>" parameter to a customized *xml2ucdb.ini* file (see ["Parameters Controlling the Inclusion of Data"](#))
 - o adding the "-excludetags <tags>" to the [xml2ucdb](#) command

All data defined between the specified tags will be skipped by the xml2ucdb processing, and therefore excluded from the UCDB when it is imported.

By doing so, any data or information that appear between the defined XML tags is effectively ignored. This could be used to stop xml2ucdb from processing information that it is not supposed to process. For example, adding the tag name that is used to identify images will stop the data being processed by xml2ucdb. The excludetags parameter can also take a list of tags for multiple tag names to be ignored in the source XML.

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 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	53
Example of a Nested Plan	54
Inheritance in Hierarchical Plans	55

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.
- Duplicate Section numbers or Titles are not allowed, and an error results when they are encountered by xml2ucdb.

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 a nested testplan inherit a parent's storing state (resulting from start/stop tags or startstoring tag) using the `-inherit` argument to the `xml2ucdb` command.

See [xml2ucdb](#) for more information. The reasons to use the `-inherit` argument, 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 does not start until a row with a specific section number (usually "1"). If the child testplan has been renumbered to (for instance: 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 is to re-define "startstoring" for the child. However, if the parent "startstoring" section had not yet been seen, the child XML file should not be imported.

The same thing happens with the XML "stoptags" and "starttags". If the parent importing a child appears between a stop and start tag, the child section should honor the parent section defined by the tags.

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

Limitations

A few limitations exist for nested testplan importation:

- The XML-type link must be the only link in the testplan section where the “child” testplan is to be imported.
- Parameter inheritance 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.

Merging Verification Plan with Test Data

Once you have captured your verification plan, the testplan and the test data must be combined in order to assess where you stand relative to your verification goals. This is achieved through the links between the sections of the verification plan and specific coverage items in the design, which is where the real power behind importing your verification plan to UCDB lies. The links are forged automatically when you merge your verification plan (top level testplan) with the coverage test data in UCDBs, taken from your test runs.

Assuming that the testplan UCDB was generated by either the Testplan Import facility or the [xml2ucdb](#) command (Refer to “[Importing an XML Verification Plan](#)”). the testplan UCDB is annotated by the import utility such that 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.

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 — Refer to “[Exporting a Plan to XML](#)”.
3. Imported .xml formatted plan into a .ucdb — Refer to “[Importing an XML Verification Plan](#)”.

4. Merge the verification plan UCDB with the test run UCDB(s) using the same methods you use to merge test data:
 - with the GUI (refer to [Merging Coverage Data](#) in the User's Manual)
 - with [vcover merge](#) (refer to “[Merging with the vcover merge Command](#)” in the User's Manual)

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

Linking Verification Plan and Coverage Items

Linking allows you to verify, via the testplan, that coverage items are being covered.

Linkable Design Constructs	58
Links Between the Plan Section and the Coverage Item	60
What are Unimplemented Links and Why Use Them?	63
Backlink Coverage Items to Plan with SV Attributes	65
Finding Unlinked Items	68
Finding Information for Missing Verification Plan Links	69
Link Debugging Using coverage tag Command	70

Linkable Design Constructs

Various constructs exist within a design, to which a testplan section can be linked. These construct types are placed in the “Type” field within the verification plan.

[Table 2-7](#) lists the constructs supported for these links.

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

Tip


 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
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

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

Coverage Construct — in “Type” field (case insensitive)	Description	Syntax
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
Formal_Proof Formal_Assumption	Formal property of assertion object	Assertions, Directives, and Generic Coverage Items
FSM	State Machine coverage scope	Code Coverage Links
Instance	All coverage on a given instance	Instances and Design Units Links
Rule	Forms a link using an automatically created virtual covergroup “UserRules” — either from a set of pre-defined Rules, or one you create.	see “ Currently Supported Rules ” and “ Rule Syntax ”
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

Links Between the Plan Section and the Coverage Item

Information regarding the links themselves, and the rules governing them are as follows.

How Links are Established	60
Multiple Items and their Links.....	61
Datafield Rules and Requirements for Links	61
Specifying Levels of Hierarchy in Plan	62

How Links are Established

The links are performed automatically when you merge an imported testplan with actual tests you have run. Key to this linking, however, is the correct association of the design constructs — various linkable constructs in the design — to the verification plan items.

The 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 a coverage construct keyword into the “Type” column/field (see [Figure 2-4](#)). [Table 2-7](#) is a complete list of the linkable coverage constructs for columns/fields.
- 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.

Figure 2-4. 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
			monitor_channel_data.FA...	CoverPoi nt	1	100
			monitor_channel_data.CRC;	CoverPoi nt	1	100

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

Multiple Items and their Links

For spreadsheets, while it is legal to have multiple items listed on a single line, separated by whitespace or a semicolon (;), it is recommended that within each section's Link entry, you list only one coverage item per line. Each line can contain a coverage item of a different type: simply match each cover item in the Link field with the correct type in the Type field.

An example of this is shown in [Figure 2-5](#). In this case, three covergroup items are listed on separate lines, and one assertion item on the fourth line.

Figure 2-5. Multi-Link and Multi-Type Example

Link	Type
INT_SRC_Cvg	CoverGroup
IPGT_Cvg	CoverGroup
PACKETLEN_Cvg	CoverGroup
assert_ignore_txbdn_write	Assertion

If all coverage items listed in the Link entry (whether on one line or multiple lines) are of one type, you can enter the Type once for that section, as shown in [Figure 2-6](#).

Figure 2-6. Multi-Link, Single-Type Example

Link	Type
INT_SRC_Cvg IPGT_Cvg PACKETLEN_Cvg	CoverGroup

Now, it is legal to list the covergroups on one line in the link column, such as: "INT_SRC_Cvg; IPGT_Cvg; PACKETLEN_Cvg;" and then have an assertion link below it. To accommodate this, you would need to enter the Type with the following format: "Covergroup;Covergroup;Covergroup;" as one line. This appears more convoluted; a single line per item is preferred for this reason.

Tip

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

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](#)” on page 227 for further details.

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

- **Link** — Required. 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** — Required. 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** — Required, unless specified in the Link. 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 might be useful in cases where there are multiple design constructs of the same name in various parts of the design, and you want the testplan section 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 2-4](#).

Related Topics

[What are Unimplemented Links and Why Use Them?](#)

[Backlink Coverage Items to Plan with SV Attributes](#)

[Finding Information for Missing Verification Plan Links](#)

[Finding Unlinked Items](#)

[Parameters Mapping Testplan Data Items](#)

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).

To get an accurate and complete picture of your coverage taking those sections into account, you can define those missing items as “Unimplemented”. This automatically generates zero-coverage, virtual objects for these items and links 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, more accurately reflecting the fact that more work is required before your design is fully covered. See: “[Counting the Coverage Contributions from Unimplemented Links](#)” on page 63.

If you do not want the missing coverage to be reflected in the overall coverage calculation, you can exclude those sections. See: “[Weighting to Exclude Testplan Sections from Coverage](#)” on page 64.

Counting the Coverage Contributions from Unimplemented Links	63
Weighting to Exclude Testplan Sections from Coverage	64

Counting the Coverage Contributions from Unimplemented Links

To ensure that your coverage count correctly accounts for incomplete or unimplemented coverage item(s) within a testplan, you can use one of two methods. Each method has certain advantages.

- `xml2ucdb -createcovitem` —

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

- Unimplemented column —

When column is added to testplan, Questa creates the specified number of empty coverage bins. This method does require the addition of a column to your testplan. However, it allows control over the number of empty bins created for individual testplan and testplan subsection links. In other words, you can correctly specify the number of empty bins to be created for each link, which helps to more accurately reflect the state of your progress toward 100% coverage.

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](#)” on page 227 for details on parameter placement.
3. Set the value in the Unimplemented column. Possible values are “No” = 0, “Yes” = 1, or <int> = multiple:
 - o Yes or “1” - creates a single empty bin
 - o No or “0” - no items created, coverage is unaffected
 - o <int> - if your item when implemented will create multiple bins, and you want to register the effect those bins would have, specify the number of coverage bins (<int>) to create.

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

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: simply set the value of the Weight column for a specified testplan section to 0 within the overall testplan.

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 of SystemVerilog attributes. The SV attributes used are part of the language itself.

Items Available for Backlinking	65
Attribute Syntax for Backlinking	65
Module and Module Instance Syntax and Examples	66
Covergroup Instance Syntax and Example	67
Notes and Limitations on Backlinking	68

Items Available for Backlinking

The following items may be linked in this manner.

- 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 two specially purposed identifiers: “tplanref” and “tplansection.”

The value of the “tplanref” identifier 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.

The "tplansection" identifier allows you to specify testplan scope hierarchy paths for backlinking. For example:

```
(* tplansection = "/testplan/mydu" *) module bottom(intf intf1);
(* tplansection = "/testplan/cvg" *) covergroup machcover @(state);
cpl: coverpoint c1 {
    option.comment = "tplansection=/testplan/mycvg/cvp1  stuff
    tplansection=/testplan/mycvg/cvp2";
}
```

The tplansection identifier may also be used within option.comment and type_option.comment of covergroups, coverpoints, and crosses.

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 Examples

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 1 — Backlinking Module Declaration

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

    . . .

endmodule
```

Example 2 — 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, such as (myinstance), 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.

Backlinking Covergroups and Coverpoints Example

```
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.

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, choose **Verification Tracker > Find Unlinked** or in the Verification Tracker window: right-click and choose **> Find Unlinked > Testplan Section**.

- From the command line, enter:
coverage unlinked -r -plansection

This displays all unlinked items.

Related Topics

[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.

Process

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 2-7](#). 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 2-7. Finding Link Information for Verification Plan

XML Import Hint						
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

- [Finding Unlinked Items](#)
- [Links Between the Plan Section and the Coverage Item](#)
- [Syntax for Links to the Coverage Model](#)
- [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	72
All-Purpose coverage tag Link	73
Functional Coverage Links	74
Code Coverage Links	77
Bin Links	79
Instances and Design Units Links	80
Directed Test Links through Test Records	80
Links for Classes	81
XML Links	83

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:

- Valid delimiters for coverage items listed within an entry (Link, Path, Type, and so on) are a whitespace (space, tab, newline, or carriage return) between coverage items. A semicolon ‘;’ is also accepted as a delimiter between covergroup items.
- A string entered in the link field 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. However, if the string begins with the path separator character (“/”), the string is assumed to be a hierarchical path to a specific design construct.
- 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

The “Tag” type allows you to link any coverage item that can be tagged with the coverage tag command.

When Tag is specified as the Type, the “”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 — Refer to “[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 for a Generic Tag Link

Section	Title	Description	Link	Type
1.3	Branches_of_DU _work_statemac h	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	74
Assertions, Directives, and Generic Coverage Items	76

Covergroups, Coverpoint and Crosses

You can define a linked item as a covergroup, a coverpoint, or a cross of a covergroup, and cause the coverage calculated for the complete covergroup, coverpoint or cross to be linked. These links can be defined in a number of ways which 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. Valid delimiters for entries within the Link column are newline, whitespace (including tab), and semicolon (;).

Table 2-9 details some examples of the various formats accepted for covergroups items:


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

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

- <CoverGroupType> — the type name given to the covergroup within the SystemVerilog source code.
 - If connecting to a covergroup type, ensure that every type name used within the environment is unique.
 - If this is not possible, precede the covergroup type with a hierarchical path to the type, such as "/top/dut/". Alternatively, the path information for a link can be entered 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, formal property of 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 either an “Assertion”, “Directive”, “CoverItem”, “Formal_Proof”, or “Formal_Assumption”

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. Refer to [“Calculation of Total Coverage”](#) in the User’s Manual for related details.

- Link Entry — Generic linking behavior. Refer to [“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.

Table 2-10. Assertion and CoverDirective Links

Link Target	Link	Type	Setting in Questa Add-In’s “Select Link” Dialog
Assertion	<Name of the assertion scope or its full Path, Wildcards allowed>	Assertion	The name of the assert scopes or its full path if "Include Full Path" is checked.
Cover directive	<Name of the CoverDirective scope or its full Path, Wildcards allowed>	Directive	The name of the CoverDirective scopes or its full path if "Include Full Path" is checked.
Formal proof, or a formal assumption property of an assertion	<Full Path to the formal proof or assumption.	Formal_Proof Formal_Assumption	Full path to the Formal_Proof/ Formal_Assumption.

Code Coverage Links

You can link a plan item with any coverage metric that is stored within the UCDB, including code coverage. Examples of possible links include:

- 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, Toggle and Statement types link directly to the code coverage statistics for a particular design scope. The details for these code coverage links are summarized in [Table 2-11](#).

Table 2-11. Code Coverage Link Details

Link Target	Link	Type	Notes	Setting in the Questa Add-In's "Select Link" Dialog
Branch	<Full Path to any scope in ucdb, Wildcards allowed>	Branch	Recursive	Define the name of the branch scopes, or its full path if "Include Full Path" is checked.
Condition	<Full Path to any scope in ucdb, Wildcards allowed >	Condition	Recursive	Define the name of the Condition scopes, or its full path if "Include Full Path" is checked.
Expression	<Full Path to any scope in ucdb, Wildcards allowed >	Expression	Recursive	Define the name of the Expression scopes, or its full path if "Include Full Path" is checked.
FSM	<Full Path to any scope in ucdb, Wildcards allowed >	FSM	Recursive	Define the name of the FSM scopes, or its full path if "Include Full Path" is checked.
Toggle	<Full Path to toggle scope, Wildcards allowed>	Toggle	Not Recursive	Define the name of the Toggle scopes, or its full path if "Include Full Path" is checked.
	"-path <path to any scope in the ucdb file, Wildcards allowed> -r -code t"	Tag	Recursive version of toggle link	

Table 2-11. Code Coverage Link Details (cont.)

Link Target	Link	Type	Notes	Setting in the Questa Add-In's "Select Link" Dialog
Statement	<Full path(s) to statement bin(s)> such as: /top/#stmt#* /top/*/#stmt#* /top/*/*/#stmt#* /top/*/*/*/#stmt#*	Bin	Not Recursive; must explicitly set all paths	<ol style="list-style-type: none"> 1. Enable "Include Full Path" 2. Select any scope of any type. 3. Press "show bins" button at the bottom of the add link dialog. A new dialog opens. 4. Select StatementBins from the "Select Bin type" List Box. All statements bins under the scope recursively appear in "Bin Link Value" list.
	"-code s -bins -path /top -r"	Tag	Recursive	

Links of Tag type, containing quoted strings (as in the Statement or Toggle entries in the table above), are treated as arguments to a coverage tag command. See "[All-Purpose coverage tag Link](#) for details".

Bin Links

Bin links are useful, though some information is required in order to make use of them.

The format for Bin links is:

- Type entry — Bin
- Link entry — Enter the full hierarchical path to bin(s).

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 Testplan Creation Using Excel Add-In](#)” 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.....	79
Bin Coverage	80

Identifying the Full Path to a Bin

The full path to a bin can be attained most easily using the -bins switch with 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 displayed in the Transcript is the following:

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

For information about the use of the string “auto[‘b<n>]” in the name, refer to [SystemVerilog 2009 type_option.merge_instances](#) in the User’s Manual.

6. Edit the testplan source (Excel spreadsheet, and so on) 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](#)” on page 28 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 (that is, "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, 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 did not 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 (refer to "[Test Attribute Records in the UCDB](#)" in the User's Manual). 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 UVM,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.

Class-based Coverage Example

```
1 package AHB_VIP;
2     `include uvm_coverage_1.sv
3 endpackage
4
5 Inside uvm_coverage_1.sv :
6     class myuvm_coverage extends uvm_subscriber;
7
```

```

8      covergroup ahb_cvg ( string p_name );
9          option.per_instance = 1;
10         option.name = p_name;
11         ...
12         coverpoint X
13         cross Y
14         ...
15     endgroup
16
17     function new(string nm = "", uvm_component p)
18         ahb_cvg=new(nm)
19     endfunction
20 ....
21 endclass

```

Using the class :

```

local myuvm_coverage ahb_cover;
ahb_cover = new("ahb_inst1", this);

```

The Covergroup "ahb_cvg" is defined in a class called "myuvm_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 2-12](#) can be used to successfully link to the coverage within the class.

Table 2-12. Class Example Linking

Link	Type	Path
ahb_cvg,ahb_inst1;	CoverGroup	
ahb_cvg,ahb_inst1:X;	CoverPoint	
ahb_cvg,ahb_inst1:Y;	Cross	
/AHB_VIP/myuvm_Coverage/ ahb_cvg,ahb_inst1;	CoverGroup	
ahb_cvg,ahb_inst1;	CoverGroup	/AHB_VIP/ myuvm_Coverage
/AHB_VIP/myuvm_Coverage/ ahb_cvg,ahb_inst1:X;	CoverPoint	
ahb_cvg,ahb_inst1:X;	CoverPoint	AHB_VIP/ myuvm_Coverage
/AHB_VIP/myuvm_Coverage/ ahb_cvg,ahb_inst1:Y;	Cross	
ahb_cvg,ahb_inst1:Y;	Cross	/AHB_VIP/ myuvm_Coverage

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 is not possible to nest more than one testplan in a single section.

“Path” entry — Ignored.

Rules-based Linking for Tracking Verification Requirements

Rules-based linking allows you to specify tracking requirements within your testplan, and link those requirements to scopes within the testplan. You can then view these requirements in the Tracker window and HTML reports.

What is a Rule?	84
Rules Usage	84
Coverage Calculation of Rules	84
Rule Syntax	85
Storing and Displaying Rules	86
Currently Supported Rules	87

What is a Rule?

A rule is a syntax for specifying a user requirement. A rule produces either a coverage bin object or a coverage scope object in the UCDB, which is linked with a testplan scope and displayed like any other coverage object.

Rules Usage

In order to use Rules, you must:

1. Specify a particular rule as the Link of your testplan, just as you would any other coverage object or test data record. For example, if your plan is an Excel based plan, you enter the rule into the Link column of the testplan, using the correct syntax for that specific rule. See “[Currently Supported Rules](#)” on page 87 for rules and syntax
2. Specify the Type as “Rule”, to indicate that the corresponding item defined in the Link is a rule. Like other coverage objects, a testplan may link to one or more rules.

Coverage Calculation of Rules

A rule is evaluated by the merge process after merging all the input files. At that point, the rule is said to be either bin-type or scope-type, depending on whether the rule is evaluated either to a boolean (true/false) value or to a coverage number.

- A bin-type rule is treated as a virtual covergroup bin with at_least=1, and that bin is linked to the corresponding testplan scope. Since this is a binary rule, the count stops at 1, if and when the first rule matches the requirement. So, the coverage of a testplan having a single link to a bin-type rule is either 0% or 100% covered.

- A scope-type rule translates to an empty virtual generic scope, with a coverage number associated with that scope. The coverage number, based on the number of rule matches are found, can be any real number value, within the 0 to 100 range.

Rule Syntax

The syntax of a rule is the name of the rule followed by the required number of arguments separated by a colon, ':':

<rule_name>[:<arg1>...<argN>]

Each rule may have its own set of specific arguments.

A special argument called OP (operator) defines a set of keywords for specifying the operation on its operand arguments.

OP:= <AND | OR | NOT | LT | GT | EQ | LE | GE | NE | REGEXP | PERCENTOF | ENUMERATE>

where LT (less than), GT (greater than), EQ (equal), LE (less than or equal), GE (greater than or equal), and NE (not equal), REGEXP (regular expression matching — for string-type attributes only), PERCENTOF (percent of specified value), and ENUMERATE (mapping of values separated by “,”).

An example of the ENUMERATE might be:

s1=d1, s2=d2, default=N

This means that if the value is s1, use the value d1. If the value is s2, use the value d2, and so on. For any other value use N.

Another special argument called “covtype” defines the different coverage types.

covtype:= [sbceftadg]

where s (statement), b (branch), c (condition), e (expression), f (fsm), t (toggle), a (assertion), d (cover directive), and g (covergroup) coverage.

Some rules may also have an expression. The supported syntax for an expression is:

```
expr := <(expr):AND:(expr)>
      | <(expr):OR:(expr)>
      | <NOT:(expr)>
      | <value_or_attrname:OP:value>
```

Example Syntax

An example of a rule is:

OBJECT_COVERED:5:ge:/top/cvg1/cvp1/bin1

This rule — when entered into the Link field with Type set to Rule — specifies a requirement that the coverage object `/top/cvg1/cvp1/bin1` must be covered by at least five tests. The merge process is instructed to create a virtual bin for this rule. It sets the count set of that bin to 1 if and only if the coverage object `/top/cvg1/cvp1/bin1` is covered by at least any five tests; otherwise, the count of that bin is set to 0.

Storing and Displaying Rules

During merge, covergroup bins are created for bin-type rules and generic scopes are created for scope-type rules. A virtual covergroup scope called “UserRules” is created under the “#TestplanGenericCoverage#” package for the purpose of holding all the bins created for different bin-type rules. A coverpoint is created for each rule type, and the name of the coverpoint is the rule name. Covergroup bins are created for each instance of a rule under the coverpoint created for the corresponding rule type. The name of a bin is constructed by concatenating the rule arguments.

For example, the following two rule specifications:

OBJECT_COVERED:5:ge:/top/cvg1/cvp1/bin1

OBJECT_COVERED:3:gt:/top/cvg1/cvp1/bin2

forms the following virtual covergroup hierarchy:

```
covergroup UserRules {  
  coverpoint OBJECT_COVERED {  
    bin _top_cvg1_cvp1_bin1_ge_5  
    bin _top_cvg1_cvp1_bin2_gt_3  
  }  
};
```

Generic scopes are created for scope-type rules under the same top-level package “#TestplanGenericCoverage#”, and the coverage value is stored in that generic scope as an attribute. Each rule instance creates a separate generic scope in UCDB. These generic scopes are displayed like virtual scopes in GUI.

Currently Supported Rules

This section lists the currently supported rules.

TEST_ATTRIBUTE	88
GLOBAL_ATTRIBUTE.....	90
OBJECT_ATTRIBUTE	91
OBJECT_COVERED	92
COVERAGE	93
FORMAL_ATTRIBUTE	95

TEST_ATTRIBUTE

This rule is for checking test attribute values. It performs two different kinds of checks, depending on which of the possible two arguments (SPECIFIC_TESTS or IN_NUM_TESTS) is used.

Syntax

TEST_ATTRIBUTE:SPECIFIC_TESTS:<spec_args>

TEST_ATTRIBUTE:IN_NUM_TESTS:<num_args>

Arguments

- SPECIFIC_TESTS
<testname>:<attribute_name>:<OP>:<value_or_enum_map>
- IN_NUM_TESTS
<num_tests>:<OP>:<attrname>:<OP>:<attrval>
- OP
<AND | OR | NOT | LT | GT | EQ | LE | GE | NE | REGEXP | PERCENTOF | ENUMERATE>

Examples

SPECIFIC_TESTS

- TEST_ATTRIBUTE:SPECIFIC_TESTS:sim100:TESTSTATUS:GT:1
Checks whether the TESTSTATUS attribute of 'sim100' test is greater than 1 or not. This rule creates a bin and the coverage of that bin is 100% if the expression satisfied otherwise the coverage of the bin is 0%.
- TEST_ATTRIBUTE:SPECIFIC_TESTS:sim100:TESTSTATUS:ENUMERATE:0=100,1=80,2=50,default=0
It checks the value of TESTSTATUS attribute of 'sim100' test and create a generic scope in UCDB for that. The coverage of the generic scope is:
 - 100% if the value of the TESTSTATUS is 0
 - 80% if the value of the TESTSTATUS is 1
 - 50% if the value of the TESTSTATUS is 2
 - 0% for all other TESTSTATUS values
- TEST_ATTRIBUTE:SPECIFIC_TESTS:sim100:myval:PERCENTOF:40
Checks the 'myval' attribute of the 'sim100' test and creates a generic scope whose coverage is the attribute's value percent of 40. That means if the attribute value is 20 then the coverage is 50%, for example.

IN_NUM_TESTS

- TEST_ATTRIBUTE: IN_NUM_TESTS:5:EQ:TESTSTATUS:GE:3

Checks whether at most 5 tests have failed (TESTSTATUS > 3).

GLOBAL_ATTRIBUTE

This rule is for checking global attribute values. It creates a scope type rule if the OP is either PERCENTOF or ENUMERATE, otherwise it creates a bin type rule.

Syntax

GLOBAL_ATTRIBUTE:<attrname>:<OP>:<attrval>

This rule gets the value of the specified global attribute and performs the operation specified by the OP argument.

Arguments

- <attrname>
Attribute name
- OP
<AND | OR | NOT | LT | GT | EQ | LE | GE | NE | REGEXP | PERCENTOF | ENUMERATE>

Examples

- GLOBAL_ATTRIBUTE:numbugs:LT:300
Checks whether the value of 'numbugs' global attribute is less than 300 or not. Creates a bin for this rule whose coverage is 100% if the value of 'numbugs' is less than 300, otherwise the coverage of the bin is 0%.
- GLOBAL_ATTRIBUTE:mycov:PERCENTOF:70
Gets the value of 'mycov' global attribute and evaluates what percent of 70 that value is. The evaluated value is then used as the coverage value of the generic scope created for this rule.

OBJECT_ATTRIBUTE

This rule is for checking user attribute values. It creates a scope type rule if the OP is either PERCENTOF or ENUMERATE, otherwise it creates a bin type rule. It operates the same as GLOBAL_ATTRIBUTE except for the object path required to specify the object of interest.

Syntax

OBJECT_ATTRIBUTE:<attrname>:<OP>:<attrval>:<objpath>

This rule gets the value of the specified scope or bin object and performs the operation specified by the OP argument

Arguments

- <attrname>
Attribute name
- OP
<AND | OR | NOT | LT | GT | EQ | LE | GE | NE | REGEXP | PERCENTOF | ENUMERATE>

Examples

- OBJECT_ATTRIBUTE:numbugs:LT:300:/top/dut/myobj
Checks whether the value of 'numbugs' attribute of /top/dut/myobj object is less than 300 or not. Creates a bin for this rule whose coverage is 100% if the value of 'numbugs' is less than 300, otherwise the coverage of the bin is 0%
- OBJECT_ATTRIBUTE:mycov:PERCENTOF:70:/top/dut/myassert
Gets the value of 'mycov' attribute of /top/dut/myassert object and evaluates what percent of 70 that value is. The evaluated value is then used as the coverage value of the generic scope created for this rule.

OBJECT_COVERED

This bin-type rule is for checking whether the specified object is covered by the number of tests as specified by the `num_tests` and the operator argument.

Syntax

`OBJECT_COVERED:<num_tests>:<OP>:<objpath`

Arguments

- `OP`
< LT | GT | EQ | LE | GE | NE | REGEXP | PERCENTOF | ENUMERATE >
Only the "EQ" operator is allowed when -1 is specified for the number of tests (<num_tests>).
- <num_tests>
Either a positive integer or -1. The -1 indicates all tests.
- <objpath>
The path to the covered object.

Examples

- `OBJECT_COVERED:3:GE:/top/mycvg/mycvp/bin1`
Checks whether the covergroup bin /top/mycvg/mycvp/bin1 is covered by at least any three tests.
- `OBJECT_COVERED:-1:EQ:/top/mycvg/mycvp/bin2`
Checks whether the bin /top/mycvg/mycvp/bin2 is covered by all the tests.
- `OBJECT_COVERED:0:LE:/top/mycvg/mycvp/illegal_bin1`
Checks whether the illegal bin /top/mycvg/mycvp/illegal_bin1 is not hit by any test.

COVERAGE

This bin-type rule is for checking coverage numbers of design units and instance scopes. It has multiple arguments to track different type of coverage numbers:

Syntax

COVERAGE:<LOCAL|RECURSIVE>:<DU|INSTANCE>:<TOTAL>:<covtype>:<objpath>

COVERAGE:<LOCAL|RECURSIVE>:<DU|INSTANCE>:<REACHED_GOAL>:<GOAL>:
 <OP>:<covtype>:<objpath>

COVERAGE:<LOCAL|RECURSIVE>:<DU|INSTANCE>:<PERCENT_OF_GOAL>:<GOAL>:
 <L>:<covtype>:<objpath>

Arguments

Coverage Arguments

- **RECURSIVE**
 Specifies the recursive coverage of the specified object
- **LOCAL**
 Specifies the local coverage of the specified object

Object Type Arguments

- **DU**
 Object is a design unit scope
- **INSTANCE**
 Object is an instance of a design unit scope

Measurement Arguments

- **TOTAL**
 Specifies the coverage number. A generic scope is created for this scope type rule and the coverage of that generic scope is same as the coverage of the specified object.
- **REACHED_GOAL**
 Specifies a check to see whether the coverage number satisfies the condition specified by OP and GOAL arguments. This measurement causes a bin type rule. The coverage of the generated bin is 100% if the coverage of the object satisfies the condition with goal otherwise the coverage of the bin is 0%.
- **PERCENT_OF_GOAL**
 Evaluates what percent of the goal the coverage of the object is. That evaluated number is used as the coverage of the created generic scope for this scope type rule.
 The num_tests argument is either a positive integer or -1. The -1 indicates all tests; only the "EQ" operator is allowed when -1 is specified for the number of tests.

Examples

- **COVERAGE:LOCAL:DU:TOTAL:sb:work.top**

Calculates the local coverage of the design unit 'work.top' considering the statement and branch coverage types only. A generic scope is created for this scope type rule and the calculated coverage is assigned to the coverage of the generic object.

- **COVERAGE:RECURSIVE:INSTANCE:REACHED_GOAL:70:GT::/top/dut**

Calculates the recursive coverage of the instance /top/dut considering all coverage types, and checks whether that coverage number is greater than 70 or not. A bin is created for this bin-type rule and 100% is assigned to the coverage of that bin if the calculated coverage of the instance is greater than 70%. Otherwise 0% is assigned to the coverage that bin.

- **COVERAGE:LOCAL:DU:PERCENT_OF_GOAL:50:g:work.mydut**

Calculates the local coverage of the design unit 'work.mydut' considering covergroup coverage only, then evaluates what percent of 50 that value is. A generic scope is created for this scope type rule and the evaluated value is assigned to the coverage of the generic object.

FORMAL_ATTRIBUTE

This rule is for checking formal attributes, formal proofs and formal assumptions, in specific or numbered tests.

Syntax

FORMAL_ATTRIBUTE:SPECIFIC_TESTS:TESTNAME:ATTRNAME:<OP>:<value>

FORMAL_ATTRIBUTE:IN_NUM_TESTS:NUMTESTS:<LT|GT|EQ|LE|GE|NE>:ATTR
/NAME:<OP>:VALUE:PROPNAME

Arguments

- OP
<AND | OR | NOT | LT | GT | EQ | LE | GE | NE | REGEXP | PERCENTOF |
ENUMERATE>
- ENUMERATE for formal status:
 - 0 = NONE, /* No formal info (default) */
 - 1 = FAILURE, /* Fails */
 - 2 = PROOF, /* Proven to never fail */
 - 3 = VACUOUS, /* Assertion is vacuous as defined by the assertion language */
 - 4 = INCONCLUSIVE, /* Proof failed to complete */
 - 5 = ASSUMPTION, /* Assertion is an assume */
 - 6 = CONFLICT /* Data merge conflict */

Alias forms of FORMAL_ATTRIBUTE

- Aliases
The rules FORMAL_PROOF, FORMAL_BOUNDED_PROOF, and FORMAL_ASSUMPTION equate to specific usages of the FORMAL_ATTRIBUTE.

```
FORMAL_PROOF:<property_name>
```

is an alias to this:

```
FORMAL_ATTRIBUTE:IN_NUM_TESTS:0:GT:#formal_status_attribute#:EQ:2:<proper  
ty_name>
```

```
FORMAL_BOUNDED_PROOF:<property_name>
```

is an alias to this:

```
FORMAL_ATTRIBUTE:IN_NUM_TESTS:0:GT:(#formal_status_attribute#:EQ:2):OR(((  
#formal_status_attribute#:EQ:4):AND:(#formal_radius_attribute#:GE:radius)):
```

```
FORMAL_ASSUMPTION:<property_name>
```

is an alias to this:

FORMAL_ATTRIBUTE:IN_NUM_TESTS:0:GT:#formal_status_attribute#:EQ:5:<property_name>

Examples

- FORMAL_ATTRIBUTE:IN_NUM_TESTS:0:GT:#formal_status_attribute#:EQ:4:<property_name>

Links to an inconclusive proof with a value of 4.

- FORMAL_ATTRIBUTE:IN_NUM_TESTS:0:GT:#formal_radius_attribute#:GE:10:/wb_arbiter_tb/X1/arb_round_robin

Links to the value of any other formal attribute, if radius is more than 10.

XML to UCDB Testplan 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 **97**

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:

Figure 2-8. 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></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>
    ...
  </chapter>
  ...
</book>
```

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

Verification Management Windows

There are two main windows for analyzing verification management results in the GUI:

- [Verification Tracker Window](#) (refer to the GUI Reference Manual) — Used for test traceability analysis. This window has a feature for user-customizable column headings. It requires a “qvman” license to view.
- [Verification Trender Window](#) (refer to the GUI Reference Manual) — Used for analyzing coverage trends in verification. This window has a feature for user-customizable column headings. It requires a “qvman” license to view.

Chapter 3

Questa Testplan Creation Using OpenOffice/ LibreOffice Calc Extension

The Questa Testplan Creation OpenOffice/LibreOffice Calc extension aids Questa Linux^{®1}/ Unix users in the creation of testplans for Questa Verification Management by assisting with the creation and capture of the testplan spreadsheet, which is then 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 Calc Extension”](#).

For reference information on the OpenOffice/LibreOffice Calc extension GUI, see [“GUI Reference for the Questa Calc Extension”](#).

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 Questa Calc Extension	101
Disabling the Questa Calc Extension	103
Re-Enabling a Disabled Questa Calc Extension	104
GUI Reference for the Questa Calc Extension	105
Using the Questa Calc Extension	112
Validation Checks Performed.	121

Installing the Questa Calc Extension

The installation package, *QuestaOOCalcAddOn_x86_64.zip*, can be found in the `<install_dir>/vm_src` directory for both Unix and Windows versions of Questa.

1. Linux[®] is a registered trademark of Linus Torvalds in the U.S. and other countries.

The Questa OpenOffice/LibreOffice extension works only on Unix/Linux 64-bit versions. So, Windows users of Questa who want to use OpenOffice/LibreOffice must copy the package to Linux/Unix for their work. Alternatively, Windows users can use the Questa Excel extension (see “Questa Testplan Creation Using Excel Add-In”).

To Install Questa OpenOffice/LibreOffice Calc Add-on, follow the steps given below.

Prerequisites

- The extension for OpenOffice is supported only in OpenOffice 3.1 and later versions.
- The extension for LibreOffice is supported only in LibreOffice 3.6.2 and later versions.

Procedure

1. Unzip the following file to your home directory:

`<questa_install_directory>/vm_src/QuestaOOCalcAddOn_x86_64.zip`

with a command such as:

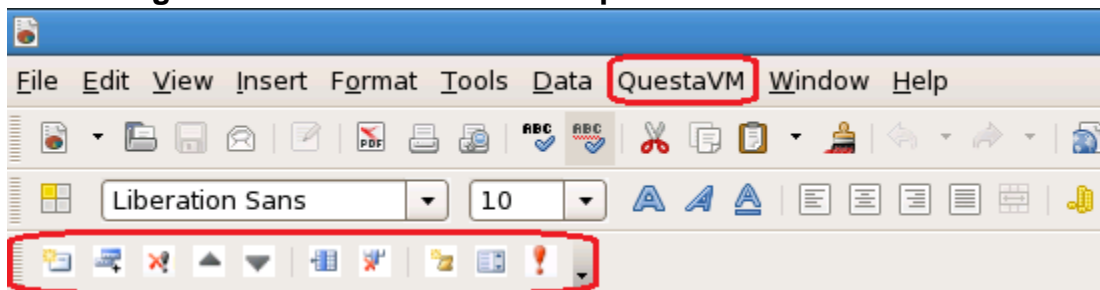
```
unzip -d <your_home_dir> $MTI_HOME/vm_src/QuestaOOCalcAddOn_x86_64
```

2. Open OpenOffice/LibreOffice Calc.
3. From the OpenOffice/LibreOffice Calc toolbar, choose **Tools > Extension Manager**. The Add Extension(s) dialog appears.
4. In the dialog, press add and select:
 - 'QuestaOpenOfficeCalcAddOn_3.oxt' for OpenOffice 3
 - 'QuestaOpenOfficeCalcAddOn_4.oxt' for OpenOffice 4
 - 'QuestaOpenOfficeCalcAddOn_3.oxt' for LibreOffice 3.x and LibreOffice 4.x.from your home directory.
5. Close both the Extension Manager dialog and the OpenOffice/LibreOffice Calc program.
6. Reopen OpenOffice/LibreOffice calc.

Results

If the extension has been successfully installed, a new menu is added to the menu bar – named 'QuestaVM' – and a new toolbar is added as shown in [Figure 3-1](#).

Figure 3-1. Questa Toolbar in OpenOffice/LibreOffice Calc



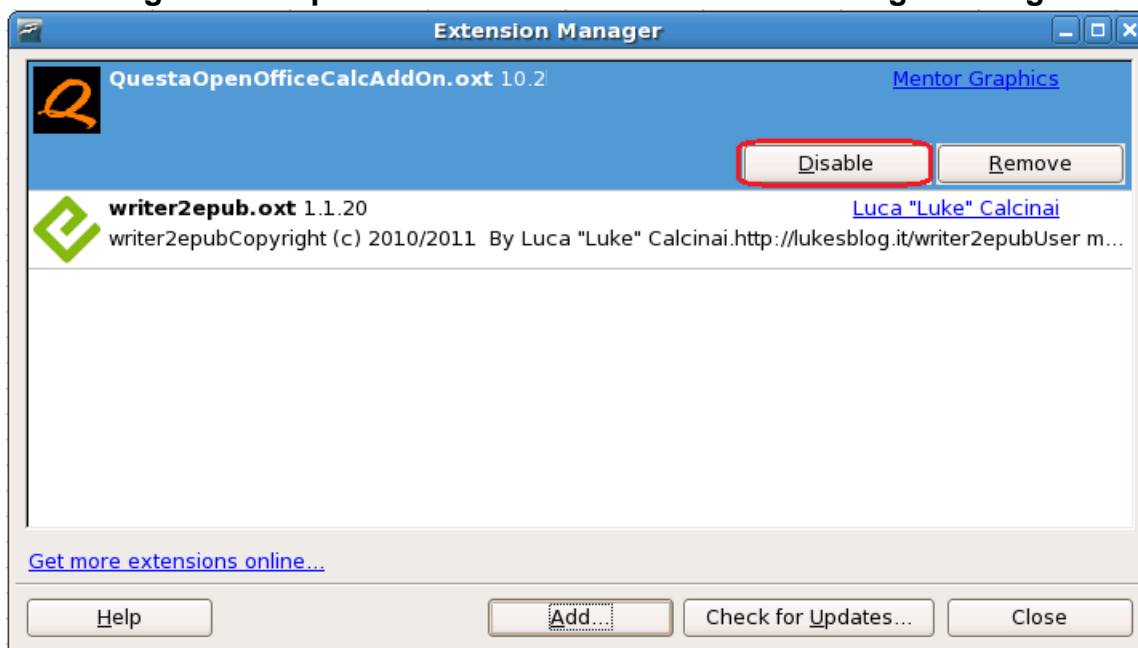
Disabling the Questa Calc Extension

If you want to disable the extension within OpenOffice/LibreOffice, without removing the installation, follow the instructions in this section.

Procedure

1. From the OpenOffice/LibreOffice Calc menu bar, choose **Tools > Extension Manager**. The Extension Manager dialog appears.

Figure 3-2. OpenOffice/LibreOffice Extension Manager Dialog



2. In the dialog, select QuestaOpenOfficeCalcAddOn from the list and click **Disable**.
3. Close both the Extension Manager dialog and the OpenOffice/LibreOffice Calc program.
4. Reopen OpenOffice/LibreOffice calc.

Re-Enabling a Disabled Questa Calc Extension

If your Questa extension has been disabled by OpenOffice/LibreOffice, you can re-enable it.

Perform the first two steps in [Disabling the Questa Calc Extension](#), but instead of choosing **Disable**, choose **Enable**. You will need to restart OpenOffice/LibreOffice for the change to take effect.

GUI Reference for the Questa Calc Extension

The Questa OpenOffice/LibreOffice Calc extension 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

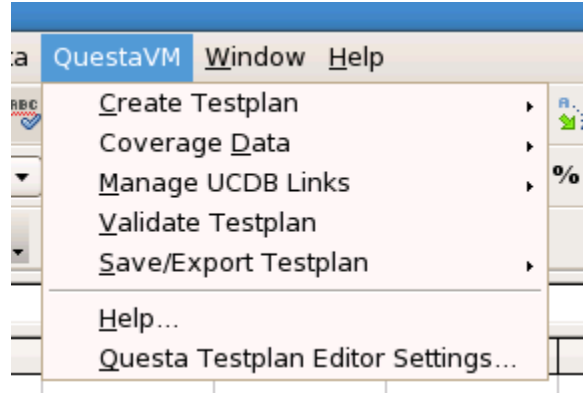
The Questa VM Menu	106
The Questa Toolbar	108
Settings Dialog	110

The Questa VM Menu

Access: **OpenOffice/LibreOffice Calc menu>QuestaVM** from the main menu

The functions that can be run from the Questa VM menu are detailed in this section.

Figure 3-3. Questa VM Menu for OpenOffice/LibreOffice Calc



Objects

Click 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 OpenOffice/LibreOffice Calc, 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 OpenOffice/LibreOffice Calc spreadsheet testplan can be saved and stored in any format that OpenOffice/LibreOffice Calc 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

Access: **OpenOffice/LibreOffice Calc** from the main menu

The Questa toolbar has seven buttons and can be opened by selecting the Questa tab in the OpenOffice/LibreOffice Calc ribbon.

Objects

- A description of the features available when clicking each icon in the Questa Toolbar is provided below.

Figure 3-4. Questa Toolbar



- Insert Section — Inserts a new plan section. Select the row at which you want to insert a new plan section and click 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 click 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 click 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 click 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 up the testplan across multiple tabs in the workbook. Hitting this button brings up a dialog box used to name the new tab, and then adds the new tab. The format for the new tab is copied from the current sheet, using the testplan section that starts one section number higher than the highest parent section number in the workbook's other tabs.
- **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. If 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 Testplan** — 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 OpenOffice/LibreOffice Calc columns where the valid information starts and ends, as well as the starting and ending (last) row in the testplan. When you click **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: OpenOffice/LibreOffice Calc's Questa Tab > Questa VM > Questa Testplan Editor Settings

Use the Questa Addon Settings dialog box 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 extension.

Figure 3-5. Questa Addon Settings Dialog Box

Questa Addon Settings

Validation

Type ☒ Interactive Validation
☐ Non-Interactive Validation

Coverage

Imported Coverage Types ☒ Functional Coverage
☒ Assertion Coverage
☒ Code Coverage

xml2ucdb

☒ Create generic coverage items for unimplemented links.
☐ Use spreadsheet title as testplan title.

Path to XML Input

Path to alternative XML2UCDB configuration file

Additional Options

Rule

Rule Syntax Version ▼

Others

☐ Allow the Add-In to Auto-Color testplans.

Objects

- 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 Types:
 - 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 in the Questa Sim Command Reference for further details):
 - Create generic coverage item for unimplemented links — when checked (as it is by default), it creates a single bin for each unlinked item (a link in the testplan which is not found in the design).
 - Use spreadsheet title as testplan title — when checked, the title for the testplan UCDB is taken from the spreadsheet title.
 - 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.
 - Path to alternative XML2UCDB configuration file — specifies the full path to the .ini file you want to use, if other than the default *xml2ucdb.ini* file.
 - Additional Options — specifies the arguments you want to apply to the xml2ucdb command.
- Rule:

This setting controls the rule syntax used in the Add Link Dialog. Make sure that you select the proper version according to the target UCDB version. The version 10.3 UCDB supports both 10.3 and 10.2 rule syntax.
- Others:

The Others area includes the **Add-In to Auto-color testplans** selection box. When checked, the Add-In toolbar buttons will color the testplan cells in the default alternating-column schema. Note that this option is checked by default.

Using the Questa Calc Extension

This section can be used as a quick start guide to using the plug-in; it assumes that the extension has already been installed.

There are three starting points to using the extension 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 OpenOffice/LibreOffice Calc spreadsheet testplan which has already been formatted to be used within Questa testplan tracking.

You can also use the extension 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	112
Generating a Testplan from a Testplan UCDB.....	115
Using an Existing Testplan as a Template.....	115
Managing Links	116
Saving and Exporting to XML and UCDB	120

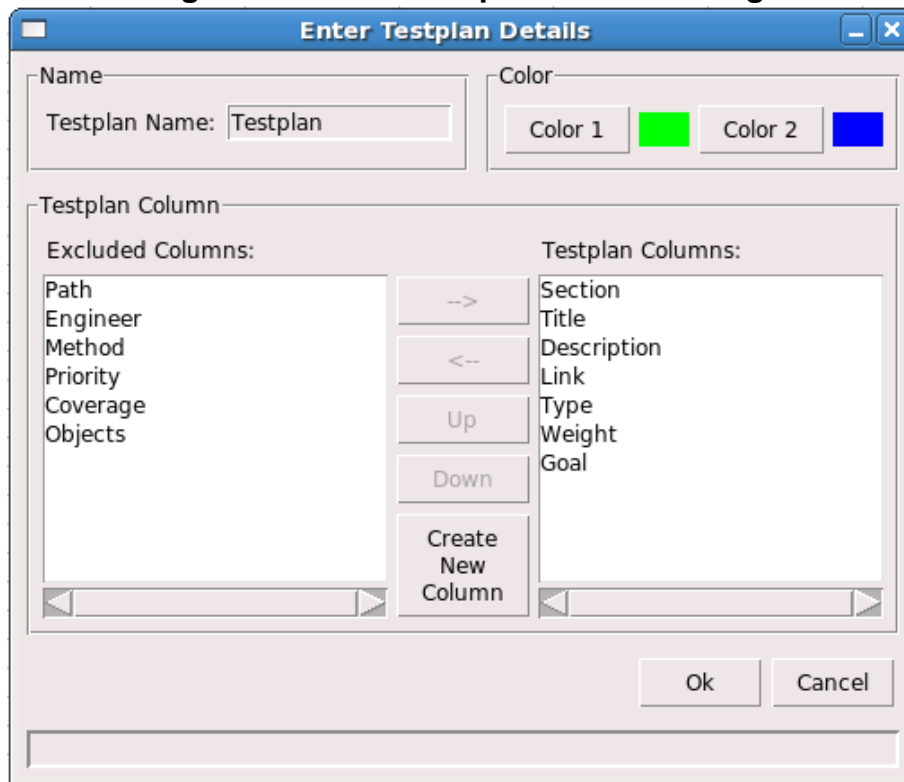
Creating a New Testplan from Scratch

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

Procedure

1. Start OpenOffice/LibreOffice Calc and go to the Questa VM menu in the extension, choose **Create Testplan > New** to display the creation dialog shown in

Figure 3-6. Enter Testplan Details Dialog.



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 want to track (such as “Milestones” or “Dates”) and have them added and ordered in the right hand list.
5. Click the **OK** button and the new spreadsheet is generated using your selections, shown below.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2														
3		#	Title	Description	Link	Type	Weight	Goal	Priority	Coverage	Objects	Milestone	Date	
4		1	Section 1				1	100						
5		1.1	SubSection 1.1				1	100						
6		2	Section 2				1	100						
7														

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 click 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 click the “Insert Sub-section” icon. This adds sub-section “1.1.1”.

Use 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 extension 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 choosing **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. choose **Validate Testplan** from the Questa VM menu. This set can be run interactive or non-interactive depending on the setting within the settings dialog box (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 OpenOffice/LibreOffice Calc supports. The extension has its own export function to generate files for use with Questa. You can choose 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 extension, and it has the added advantage of knowing that all the functions of the extension are compatible with the generated format.

To generate a testplan from a testplan UCDB, use this procedure.

Procedure

1. Choose **Create Testplan > from Testplan UCDB** from 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 opens a file browser to choose the UCDB file to use for the generation.
3. Find the UCDB file you want to use, and click **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 Testplan as a Template

As long as an existing spreadsheet follows a similar format to the format supported within Questa, the extension 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 extension, providing that the format of the testplan is consistent with Questa's OpenOffice/LibreOffice Calc template. The extension 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 does not 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

Managing links is a central component of a testplan management.

How the Links Work	116
Adding Links with the Select Link Dialog.....	116
Removing Existing Association with Testplan	119

How the Links Work

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 extension generates cache link files from the UCDB and stores them in a tmp directory named:

`<OO_install_dir>/QuestaOOAddOnCache`

These files are updated by the extension 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

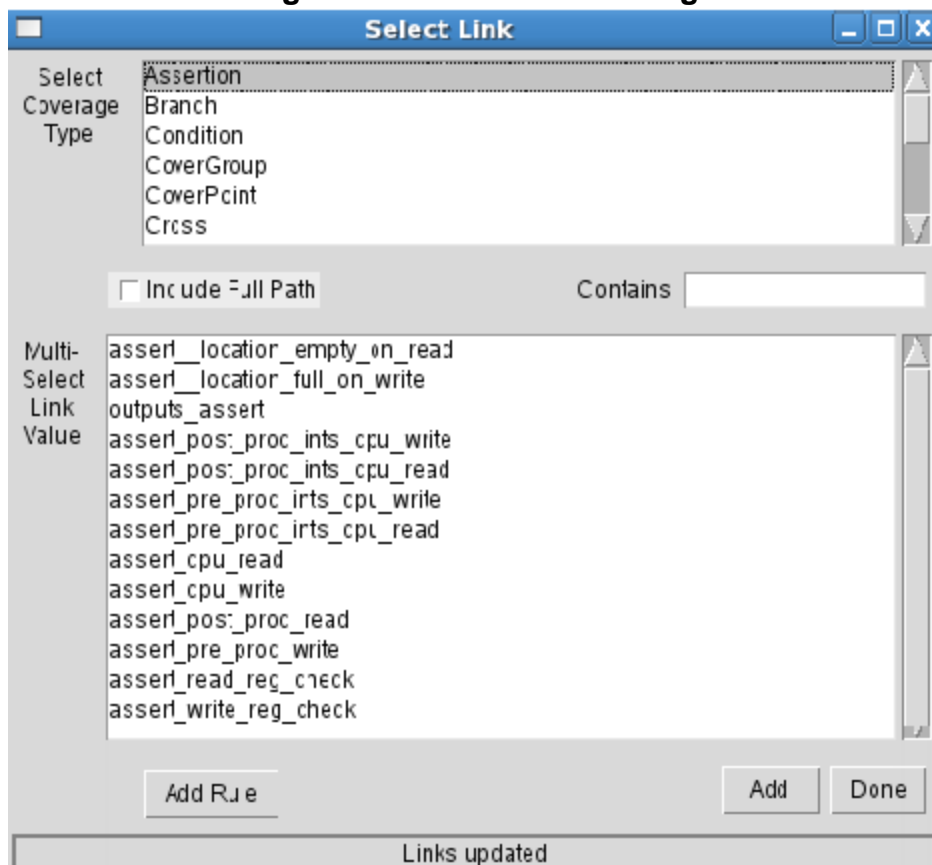
Links can be added to the plan from the OpenOffice add-in.

Add links to the selected cell in a spreadsheet using the Select Links dialog, as follows.

Procedure

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 3-7. Select Link Dialog



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 add a rule:
 - a. If applicable, select the scope (Assertion, Test, Instance, DU, ...) related to the rule.

Figure 3-8. Adding a Rule with Select Link

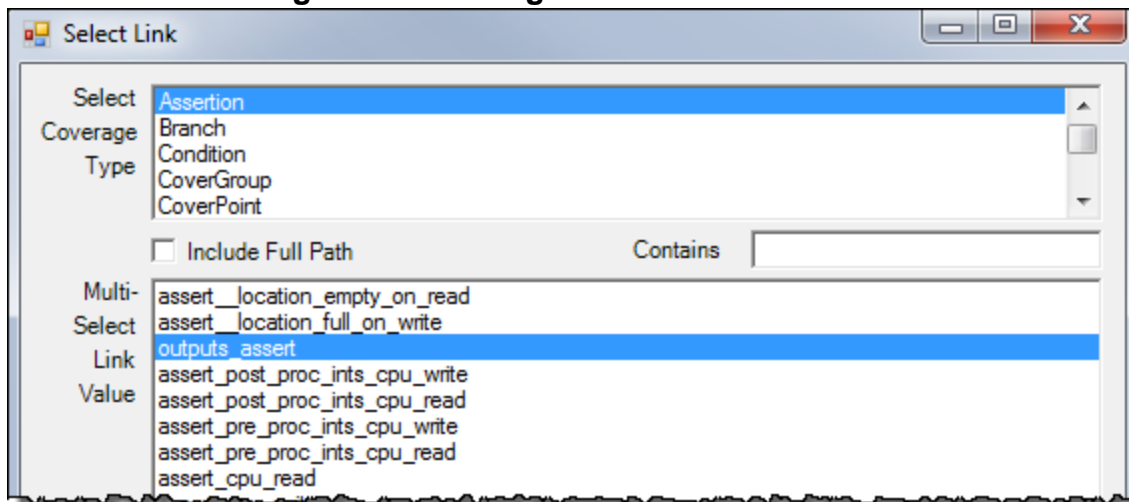
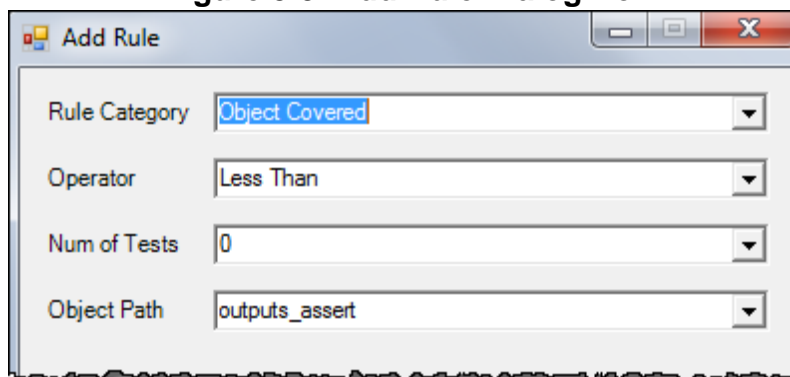


Figure 3-8 shows that the “Assertion” Coverage Type and “outputs_assert” Link Value are selected.

- b. Press the **Add Rule** button. The Add Rule dialog appears as shown in Figure 3-9.

Figure 3-9. Add Rule Dialog Box



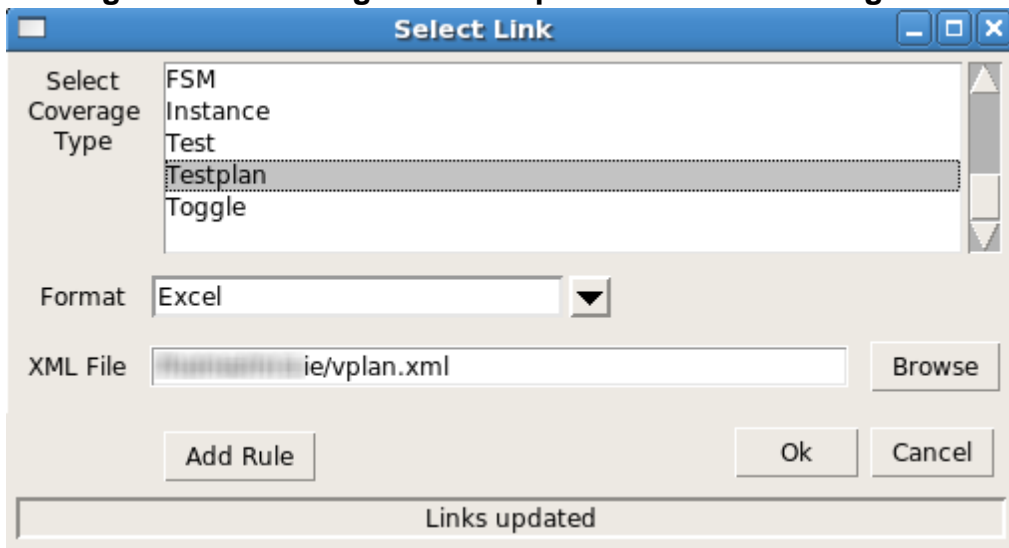
Because a Link Value was selected in the Select Link dialog, when the Add Rule dialog opens, the Objects Path field is automatically populated with that value.

- c. Select the Rule Category and fill in the other rule parameters.
- d. Select **Add** to add the rule to the Cell.

See “[Settings Dialog](#)” for details on setting up the rule syntax version. For more details on using Rules for linking coverage items to the testplan, see “Rules-based Linking for Tracking Verification Requirements”.

6. To add a sub-testplan:
 - a. Select Testplan from the Select Coverage Type List as shown in [Figure 3-10](#).

Figure 3-10. Adding a Sub-testplan with Select Dialog Box



- b. Select the format of the xml file from the Format pulldown menu.
- c. Press Browse button and select your XML File.

If the start root of the new sub-testplan is the same as the section number, you do not need to add the '-startroot' option.

- 7. Select the **Add** button in the dialog to add the links and types into the cells of the selected row. **Add** also dismisses the dialog.

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 the following sections.

How and Why to Export TestPlan to XML..... 120

How and Why to Export Testplan to UCDB..... 120

How and Why to Export TestPlan to XML

Choosing **Save/Export Testplan > to XML** from the main menu exports an XML file that can import the testplan into Questa using the xml2ucdb program.

The XML is output in a format that can be processed using the OpenOffice/LibreOffice Calc 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 does not need to be saved as XML, it can be kept as an OpenOffice/LibreOffice Calc binary file, and the output from this process can be used to import the data to UCDB (through xml2ucdb).

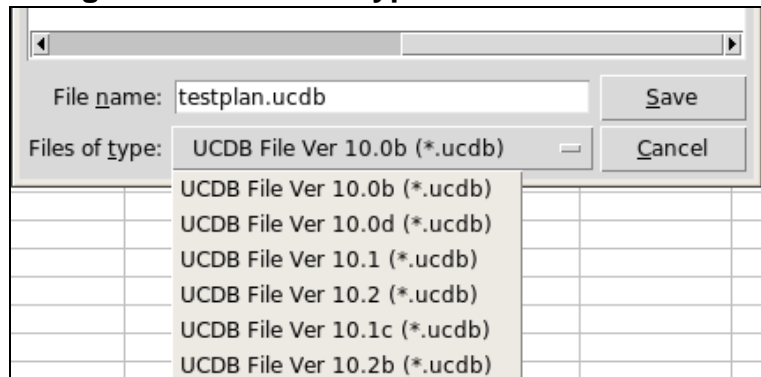
How and Why to Export Testplan to UCDB

By choosing **Save/Export Testplan > to UCDB** from the main menu selection, you not only export an XML file, but you also generate a script to run the xml2ucdb executable and then run that 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 10.3 UCDB and try to open it with Questa 10.2, it will not work. Take care to ensure you sure to select the correct version of the UCDB for OpenOffice/LibreOffice Calc to use in the **Files of type:** pull-down menu, within the **Save As** dialog box.

Figure 3-11. Files of type: UCDB File Version



- **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 does not 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 validation process performs the following sequence of checks:

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.

When the validation process is complete, a message box is displayed.

Chapter 4

Questa Testplan Creation Using Excel Add-In

The Questa Testplan Tracking Excel Add-In aids Questa Windows users in the creation of testplans for Questa Verification Management by assisting with the creation and capture of the testplan spreadsheet. The spreadsheet is then 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	123
Disabling the Questa Excel Add-In	125
Re-Enabling a Disabled Questa Add-In	126
GUI Reference for the Questa Add-In	127
Using the Questa Add-In	135
Validation Checks Performed	145

Installing the Excel Add-In

This section contains the directions for installation.

Restrictions and Limitations

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

OpenOffice Calc Plug-In (see “Questa Testplan Creation Using OpenOffice/LibreOffice Calc Extension.”)

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


Prerequisites

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 2007, 2010, or 2013.
- 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.

Procedure

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

Tip
 (Windows Vista) Consult the Prerequisite section for necessary preliminary steps.

2. Run the installer. The self extracting installer, *QuestaExcelAddIn.msi*, can be found in the `<Questa_install_dir>/vm_src` directory for both Unix and Windows versions of Questa:

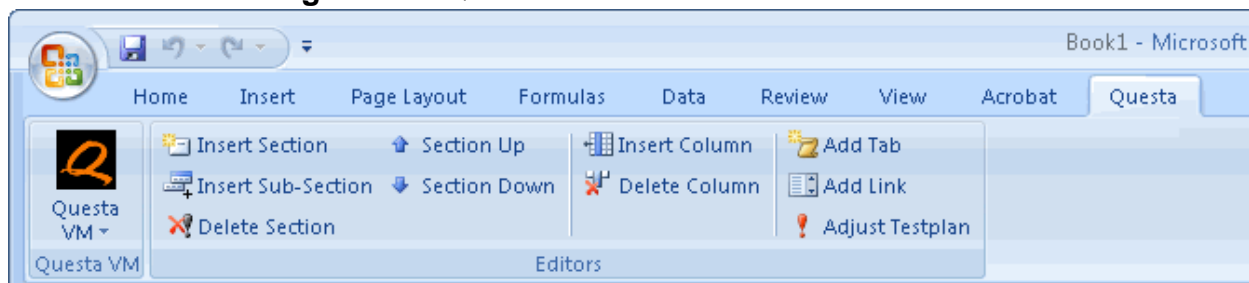
`<questa_install_dir>/vm_src/QuestaExcelAddIn.msi`

3. Follow the instructions that appear in the Setup Wizard.
4. The Installer adds all necessary files for the Add-In functionality and makes the necessary registry changes to integrate the Add-In with Excel.

Results

If the Add-In has been successfully installed, a new tab, called **Questa**, is added to the Excel ribbon. The tab includes two groups: QuestaVm and Editors Group, as shown in [Figure 4-1](#).

Figure 4-1. Questa Tab in Excel 2007/2010/2013



If you do not see the Questa menu/tab, it is possible that the Questa add-in was disabled. If that is the case, even if you installed the add-in, it would fail to appear. See “[Re-Enabling a Disabled Questa Add-In](#)” for instructions on re-enabling a disabled item.

For other issues, consult the following document for possible solutions:

`<Excel_install_path>/vm_src/AfterInstallReadMe.txt`

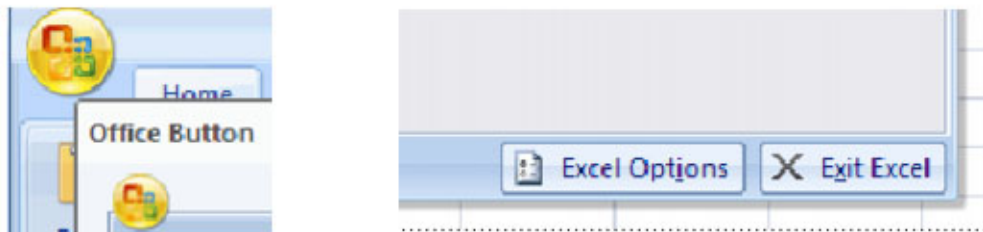
Disabling the Questa Excel Add-In

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

Procedure

1. Open Microsoft Excel (2007/2010/2013) and click the Office Button.
2. Select Excel Options.

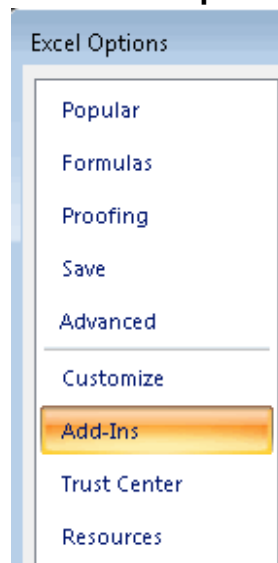
Figure 4-2. Office button and Excel Options



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

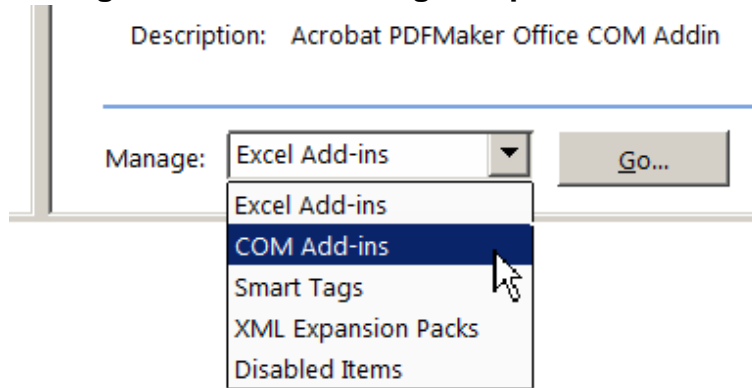
3. Select Add-Ins from the options menu.

Figure 4-3. Excel Options menu



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

Figure 4-4. Excel Manage dropdown menu



This opens a dialog entitled COM Add-Ins.

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

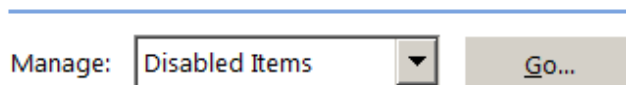
This disables the Questa Excel Add-In, removing the Questa tab from the ribbon bar.

Re-Enabling a Disabled Questa Add-In

If your Questa Add-In has been disabled by Excel, you can re-enable using the following instructions.

Procedure

1. As shown in the first three steps in [Disabling the Questa Excel Add-In](#), select the Microsoft Office button, select Excel Options, select Add-Ins.
2. From the drop-down box next to “Manage:” select “Disabled Items” and click the “Go...” button.



A dialog box for Disabled Items opens which could list the Questa Add-In as having been disabled. If it appears in the list, you can it and click the “Enable” button to re-enable the Add-In.

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

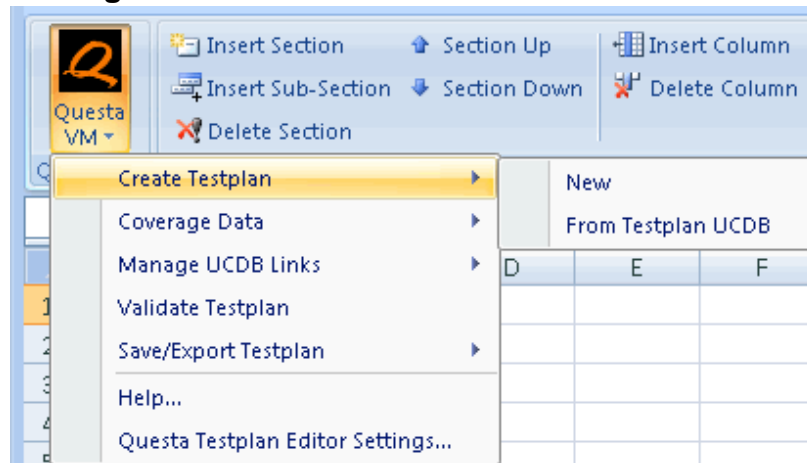
The Questa VM Menu	128
The Questa Toolbar	130
Settings Dialog	132

The Questa VM Menu

Access: **Excel** ribbon> **QuestaVM**

The functions that can be run from the Questa VM menu are detailed in this section.

Figure 4-5. Questa VM Menu for Excel Add-In



Objects

Table 4-1. 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 4-1. 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.

Usage Notes

Click the Questa VM icon to display the popup menu and select one of the options shown in the table above.

The Questa Toolbar

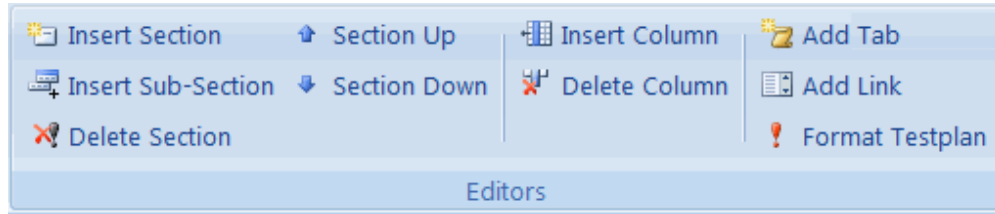
Access: **Excel** menu

The Questa toolbar has buttons which can be selected to access functionality of the AddIn. The toolbar can be opened by selecting the Questa tab in the Excel ribbon.

Objects

- Excel Add-in Functionality Options

Figure 4-6. Questa Toolbar: Features and Icons



- Insert Section — Inserts a new plan section. Select the row at which you click to insert a new plan section and click 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 click to insert a new sub-section and click 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 click 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 click 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 up the testplan across multiple tabs in the workbook. Hitting this button brings up a dialog box used to name the new tab, and then adds the new tab. The format for the new tab is copied from the current sheet, using the testplan section that starts one section number higher than the highest parent section number in the workbook's other tabs.
- 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. If you want to use a different UCDB file, you must remove the association between the UCDB file and the spreadsheet by choosing the **Manage UCDB Links > Remove** menu item, which deletes the cache workbook. See “[Managing Links](#)”.

- Format Testplan — 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 click **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

The Settings dialog box controls 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 4-7. Settings Dialog Box

Settings

Validation

Type ☒ Interactive Validation ☐ Non-Interactive Validation

Coverage

Imported ☒ Functional Coverage

Coverage ☒ Assertion Coverage

Type ☒ Code Coverage

xml2ucdb

☐ Create generic coverage item for unimplemented links.

☐ Use Spreadsheet title as testplan title.

Path to XML Input

Path to alternative XML2UCDB configuration file

Additional Options

Rule

Rule Syntax Version

Others

☒ Allow the Add-In to Auto-Size testplan columns.

☒ Allow the Add-In to Auto-Color testplans.

Objects

- Validation Types:
 - 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 Types:
 - 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 xml2ucdb command in the Questa Sim Command Reference for further details:
 - Create generic coverage item for unimplemented links — when checked (as it is by default), it creates a single bin for each unlinked item (a link in the testplan which is not found in the design).
 - Use spreadsheet title as testplan title — when checked, the title for the testplan UCDB is taken from the spreadsheet title.
 - 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.
 - Path to alternative XML2UCDB configuration file — specifies the full path to the .ini file you want to use, if other than the default *xml2ucdb.ini* file.
 - Additional Options — specifies the arguments you want to apply to the xml2ucdb command.
- Rule:

This setting controls the rule syntax used in the Add Link Dialog. Make sure that you select the proper version according to the target UCDB version. The version10.3 UCDB supports both 10.3 and 10.2 rule syntax.
- Others:
 - Add-In to Auto-Size testplan columns — When checked, the Add-In re-sizes the columns of the testplan to automatically fit Data in the column.

- Allow Add-In to Auto-Color testplans — When checked, the Add-In toolbar buttons will color the testplan cells in the default alternating-column schema. Note that this option is checked by default.

Related Topics

[How and Why to Export Testplan to UCDB](#)

[Counting the Coverage Contributions from Unimplemented Links](#)

Using the Questa Add-In

This section can be used as 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.

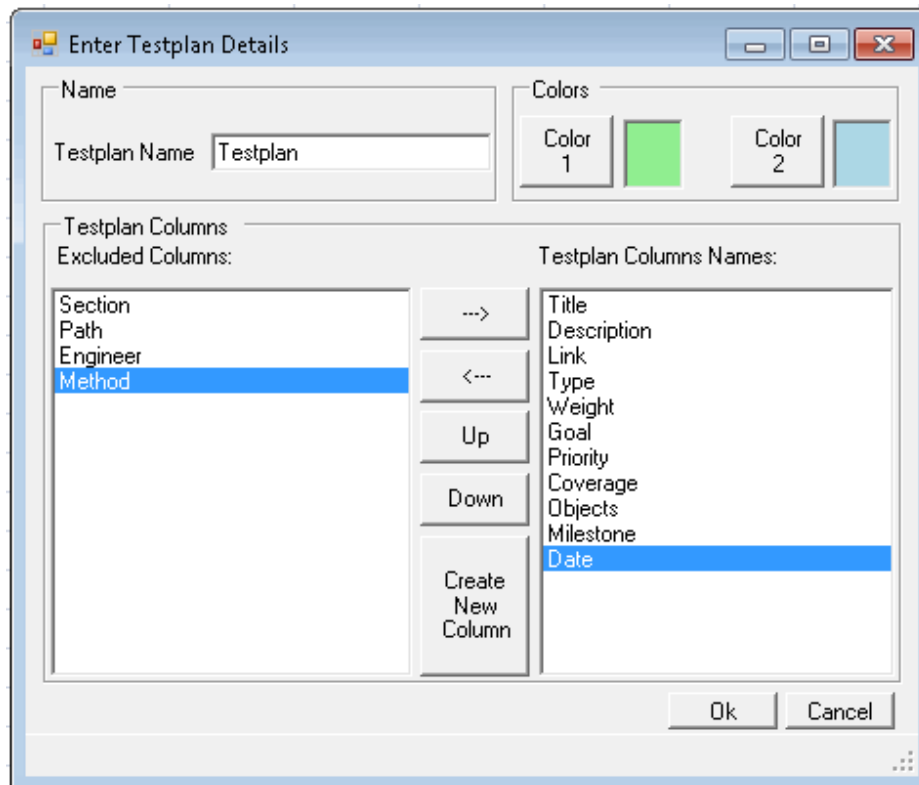
Creating a New Testplan from Scratch	136
Generating a Testplan from a Testplan UCDB.....	138
Using an Existing Excel Testplan as a Template	138
Managing Links	140
Saving and Exporting to XML and UCDB	144

Creating a New Testplan from Scratch

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

Procedure

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



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 want to track, for example “Milestones” or “Dates”, and have them added and ordered in the right hand list.

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

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2														
3		#	Title	Description	Link	Type	Weight	Goal	Priority	Coverage	Objects	Milestone	Date	
4		1	Section 1				1	100						
5		1.1	SubSection 1.1				1	100						
6		2	Section 2				1	100						
7														

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 click 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 click the “Insert Sub-section” icon. This adds sub-section “1.1.1”.

Use **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 choosing **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. Choose **Validate Testplan** from the Questa VM menu. You can run this set interactive or non-interactive, depending on the setting within the settings dialog box (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 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, follow these instructions.

Procedure

1. Choose 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 click **OK**.
This opens a file browser to choose the UCDB file to use for the generation.
3. Find the UCDB file you want to use, and click **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 does not 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

Managing links is a central component of a testplan management.

How the Links Work	140
Adding Links with the Select Link Dialog.....	140
Removing Existing Association with Testplan	143

How the Links Work

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

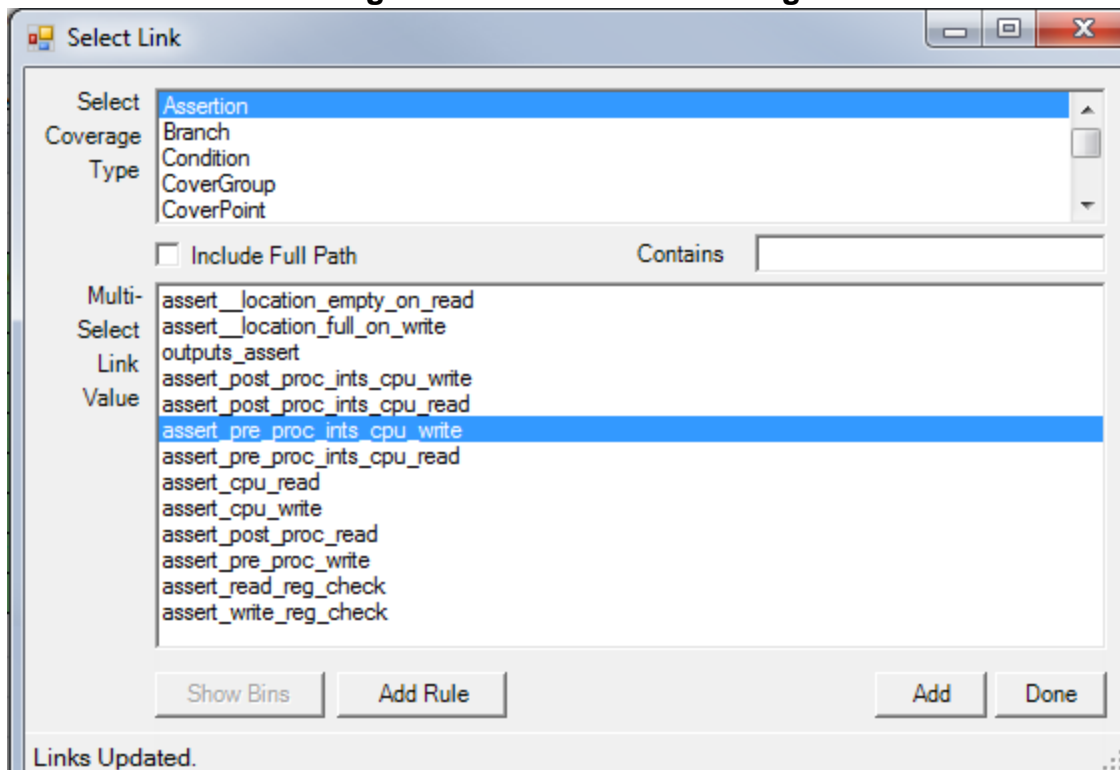
Links can be added to the plan from the Excel add-in.

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

Procedure

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 4-8. Select Links Dialog

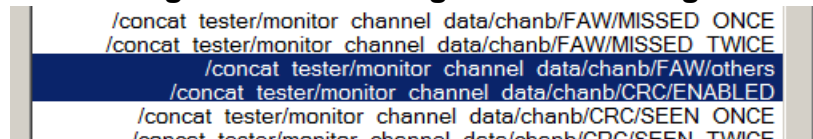


3. Select the coverage type for the link you want to add from the Select Coverage Type list.
4. Select one or more links from the Multi-Select Link Value list.
5. To see and/or link bins for the coverage type selected:
 - a. If the selected Coverage Type is anything other than **Du**, check the Include 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 inactive (gray) if the Coverage Type is **Test**.

Figure 4-9. Selecting Bins for Linking



- d. Select **Add** in the Select Bin Link dialog box.

6. To add a rule:
 - a. In applicable testplan section, select the scope (Assertion, Test, Instance, DU, ...) related to the rule.

Figure 4-10. Adding a Rule with Select Link

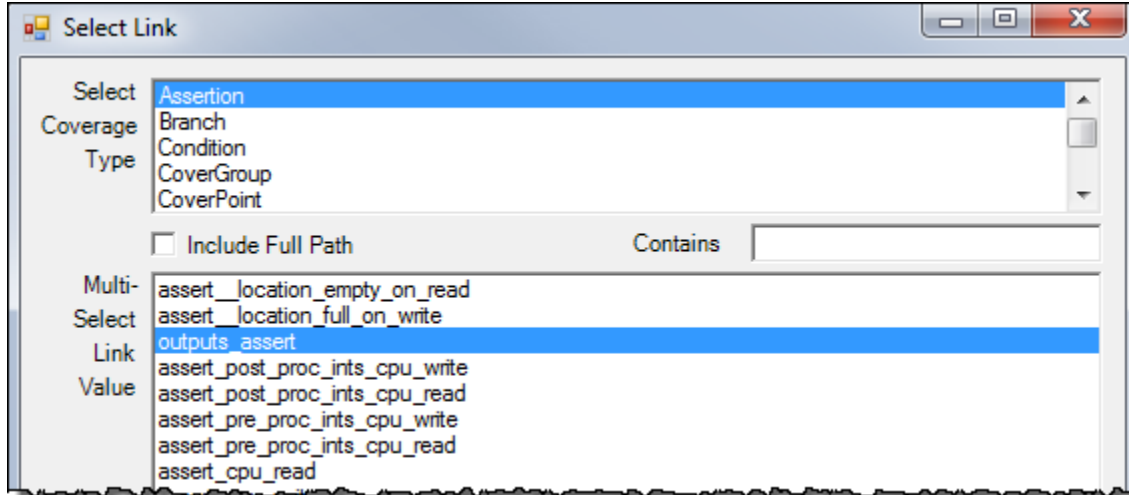
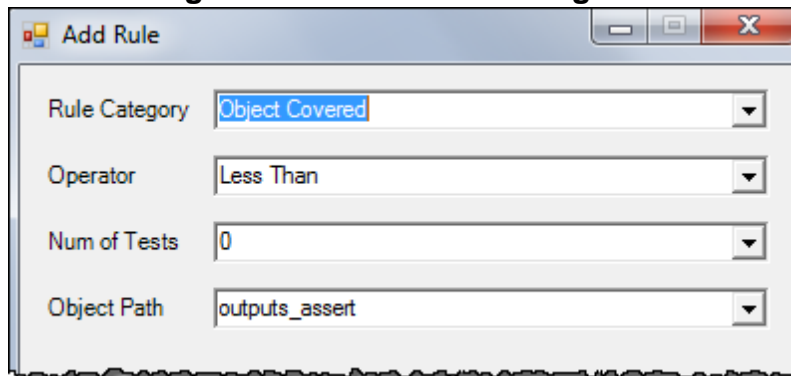


Figure 4-10 shows that the “Assertion” Coverage Type and “outputs_assert” Link Value are selected.

- b. Press the **Add Rule** button. The Add Rule dialog appears as shown in Figure 4-11.

Figure 4-11. Add Rule Dialog Box



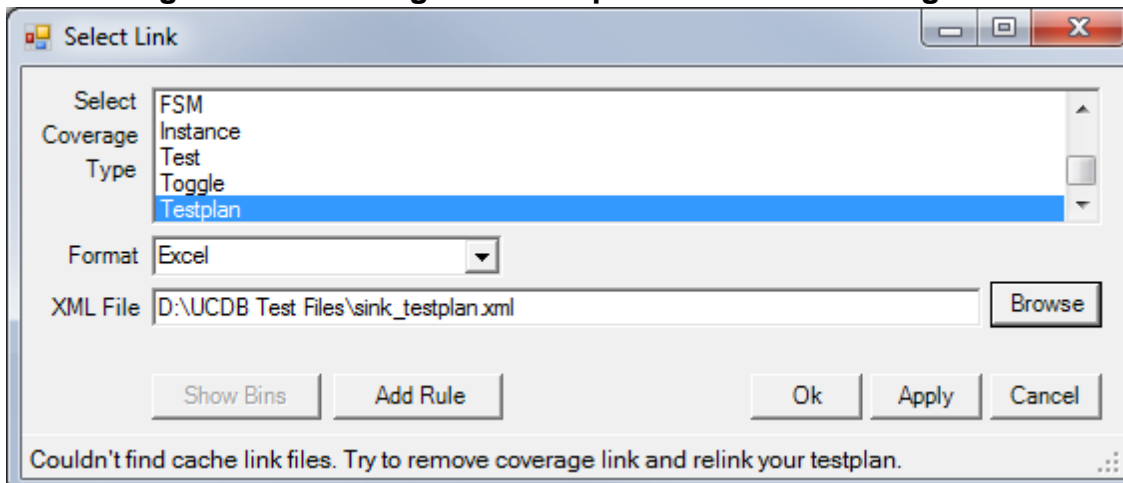
Because a Link Value was selected in the Select Link dialog, when the Add Rule dialog opens, the Objects Path field is automatically populated with that value.

- c. Select the Rule Category and fill in the other rule parameters.
 - d. Press **Add** button to add the rule to the Cell.

See “[Settings Dialog](#)” for details on setting up the rule syntax version. For details on using Rules for linking coverage items to the testplan, see “Rules-based Linking for Tracking Verification Requirements”.

7. To add a sub-testplan:
 - a. Select Testplan from the Select Coverage Type List as shown in [Figure 4-12](#).

Figure 4-12. Adding a Sub-testplan with Select Dialog Box



- b. Select the format of the xml file from the Format pulldown.
 - c. Press Browse button and select your XML File.

If the start root of the new sub-testplan is the same as the section number, you do not need to add the '-startroot' option.

8. Select the **Add** button in the dialog to add the links and types into the cells of the selected row.

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, choose:

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 explained below.

How and Why to Export TestPlan to XML..... 144

How and Why to Export Testplan to UCDB..... 144

How and Why to Export TestPlan to XML

Choosing **Save/Export Testplan > to XML** from the main menu exports an XML file that imports the testplan into Questa using 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 titled “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.

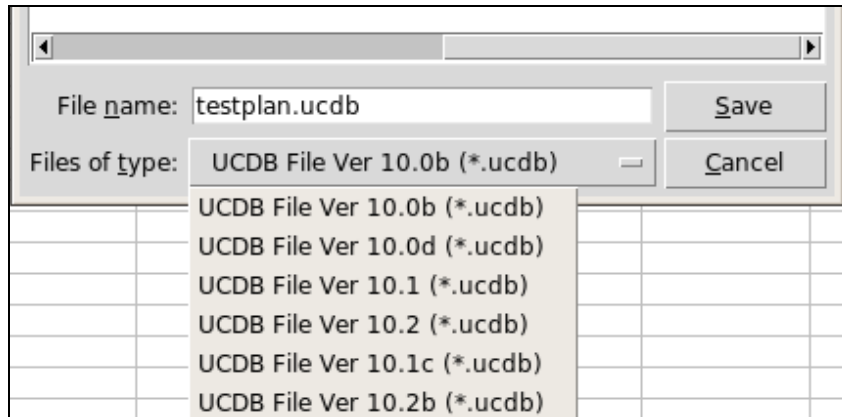
The `xml2ucdb` function transverses the testplan data within the spreadsheet and generates an XML file. The benefit of this method is that the testplan spreadsheet itself does not need to be saved as XML, it can be kept as an Excel binary file, and the output from this process can be used to import the 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 10.3 UCDB and try to open it with Questa 10.2, 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.



- **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 does not 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.

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

Chapter 5

Automated Run Management

The Verification Run Manager (VRM) accelerates the verification process through the use of automation and an understanding of the typical verification process.

The Verification Run Manager is a separate tool that you can use to manage regressions suites and multiple tools. Introductory information on run management is provided in this section, with an runnable example, for the purposes of illustrating its potential to aid your verification efforts. Details on VRM can be found in the *Questa Verification Run Management User Guide*.

Introduction to Run Management	147
Run Management Examples.....	148

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.

The verification plan 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, and so on.
- How progress will be measured - Usually in the form of functional coverage, assertion coverage, code coverage, coverage from Formal engines, and so on.

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 by allowing you to easily describe a particular regression suite. The description is called the Run Manager Database RMDB. Through it, you can create highly parameterizable and portable regression environments which can be reused in many different ways. In addition, VRM provides hooks such that you can better adapt it in your environment.

The Questa SIM Verification Run Manager can be invoked in a GUI as well as a command line mode. For details on all aspects of the VRM, refer to the “Questa Verification Run Management User Guide”.

Run Management Examples

The RMDB is “the brain” of the regression management. Contained in this section is a template/example, as well as a more complete example, which contains a working version of a RMDB.

Template Example

Figure 5-1. 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:

```
vrn -rmdb myregrxn.rmdb mygroup
```

Complete Example

To view and run a complete yet simple example of the VRM, see `<install_dir>/examples/vrm/fpu_ovm`. The RMDB for this example is shown in [Example 5-2](#).

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

Figure 5-2. 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="TESTCASE">fpu_test_patternset 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="DPI_HEADER_FILE" type="file">(%DATADIR%) dpiheader.h</parameter>
      <parameter name="REF_MODEL_CPP" >fpu_tlm_dpi_c.cpp</parameter>

      <!-- If ucdbfile parameter is defined and mergefile parameter is defined, VRM will
perform
          automatic merging of ucdb file 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>
      <parameter name="mergeoptions">-testassociated</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 (%RMDBDIR%)/
transformrule.txt</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" type="file">.. fpu vplan vplan_excel.xml</
parameter>
      <parameter name="tplanoptions" >-format Excel -startstoring 3</parameter>

      <parameter name="QUESTA_HOME" type="tcl">[file join $::env(MODEL_TECH) ..]</parameter>
      <parameter name="QUESTA_OS" type="tcl" >[file tail $::env(MODEL_TECH)]</parameter>

      <parameter name="OVM_VERSION">ovm-2.1.1</parameter>

      <parameter name="VLIB" type="file">(%QUESTA_HOME%) (%QUESTA_OS%) vlib</parameter>
      <parameter name="VMAP" type="file">(%QUESTA_HOME%) (%QUESTA_OS%) vmap</parameter>
      <parameter name="VCOM" type="file">(%QUESTA_HOME%) (%QUESTA_OS%) vcom</parameter>

```

```

    <parameter name="VLOG"          type="file">(%QUESTA_HOME%) (%QUESTA_OS%) vlog</parameter>
    <parameter name="VOPT"          type="file">(%QUESTA_HOME%) (%QUESTA_OS%) vopt</parameter>
    <parameter name="VSIM"          type="file">(%QUESTA_HOME%) (%QUESTA_OS%) vsim</parameter>
    <parameter name="XML2UCDB"      type="file">(%QUESTA_HOME%) (%QUESTA_OS%) xml2ucdb</
parameter>
    <parameter name="VCOVER"        type="file">(%QUESTA_HOME%) (%QUESTA_OS%) vcover</
parameter>

    <parameter name="DUT_LIB"       type="file">(%DATADIR%) dut_lib</parameter>
    <parameter name="WORK_LIB"      type="file">(%DATADIR%) work</parameter>
    <parameter name="OVM_LIB"       type="file">(%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="REF_SRC"   type="file">.. fpu tlm c</parameter>
        <parameter name="OVM_HOME" export="yes" type="file">(%QUESTA_HOME%) verilog_src
(%OVM_VERSION%)</parameter>
    </parameters>
    <members>
        <member>compile_init</member>
        <member>compile_elements</member>
        <member>optimize</member>
    </members>
</runnable>

<!-- Create physical libraries.
    Use vmap to map physical library to logical library name
    This will create a modelsim.ini file in the
    DATADIR area that can be shared by other runnables -->
<runnable name="compile_init" type="task">
    <execScript launch="vsim">
        <command>(%VLIB%) (%WORK_LIB%)</command>
        <command>(%VLIB%) (%DUT_LIB%)</command>
        <command>set ::env(MODELSIM) (%DATADIR%)/modelsim.ini</command>
        <command>(%VMAP%) mtiOvm (%OVM_LIB%)</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>

<!-- Using localfile element content, a symbolic link to %MODELSIM_INI% is made
      by VRM for this task runnable -->
<runnable name="compile_dut" type="task">
  <parameters>
    <parameter name="VCOM_ARGS">-2002</parameter>
    <parameter name="VOPT_ARGS">-pdu +cover=sbceft</parameter>
    <parameter name="FILELIST" type="file">vhdl.f</parameter>
  </parameters>
  <localfile type="link" src="(%DATADIR%)/modelsim.ini" />
  <execScript launch="vsim">
    <command>(%VCOM%) -work (%DUT_LIB%) (%VCOM_ARGS%) -f (%FILELIST%)</command>
    <command>(%VOPT%) -work (%DUT_LIB%) (%VOPT_ARGS%) fpu -o fpu_pdu </command>
  </execScript>
</runnable>

<!-- TB System compilation group including compilation of DPI c code -->
<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>
  </members>
</runnable>

<!-- This is an example of how to create a conditional runnable. In this example
      OVM will be compiled into the work library if parameter DO_COMPILE_OVM is set to 1
      otherwise the precompiled OVM shipped in Questa is used -->
<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="(%DATADIR%)/modelsim.ini" />
  <execScript launch="vsim">
    <command>(%VLOG%) -work (%WORK_LIB%) (%VLOG_ARGS%) -f (%FILELIST%)</command>
  </execScript>
</runnable>

<!-- Automated DPI compile flow is used.
      Vlog performs DPI c compilation automatically.
      There is no need for -sv_lib < > option to vsim in this flow -->
<runnable name="compile_ovm_tb" type="task">
  <parameters>
    <parameter name="VLOG_ARGS">-suppress 2181 -Epretty vlog.sv -dpiheader
(%DPI_HEADER_FILE%)</parameter>
    <parameter name="PLUS_ARGS">+define+SVA+SVA_DUT</parameter>
    <parameter name="PLUS_ARGSbad">(%PLUS_ARGS%)+FPU_TB_BUG</parameter>
  </parameters>
</runnable>

```

```
<parameter name="FILELIST" type="file">vlog.f</parameter>
</parameters>
<localfile type="link" src="(%DATADIR%)/modelsim.ini" />
<execScript launch="vsim">
  <command>(%VLOG%) -work (%WORK_LIB%) (%VLOG_ARGS%) (%PLUS_ARGS(%SETUP:%)%) -f
(%FILELIST%)</command>
  <command>(%VLOG%) -work (%WORK_LIB%) -ccflags " -I(%DATADIR%) " (%REF_SRC%)/
(%REF_MODEL_CPP%)</command>

</execScript>
</runnable>

<runnable name="optimize" type="task">
  <parameters>
    <parameter name="VOPT_ARGS">+acc</parameter>
  </parameters>
  <localfile type="link" src="(%DATADIR%)/modelsim.ini" />
  <execScript launch="vsim">
    <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 regression 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 enables automatic pass / fail
determination to be based on value of UCDB attribute called TESTSTATUS by 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 -msgmode both -displaymsgmode both</parameter>
<parameter name="VSIMARGS">{%VSIMSWITCHES%} -do "source {%VSIM_DOFILE%}" -lib
{%WORK_LIB%}</parameter>
<parameter name="PATTERNSET_FILENAME" type="file">../tb/golden/
pattern_set_ultra_small.pat</parameter>
<parameter name="PATTERNSET_MAXCOUNT">-1</parameter>
</parameters>
<localfile type="copy" src="scripts/wave.do"/>
<localfile type="copy" src="scripts/wave_batch.do"/>
<localfile type="link" src="{%DATADIR%}/modelsim.ini" />
<execScript launch="vsim" mintimeout="300">
  <command>if {[string match {%ITERATION%} fpu_test_patternset]} {</command>
  <command> if {[file exist {%PATTERNSET_FILENAME%}]} {</command>
  <command> file copy -force {%PATTERNSET_FILENAME%} .</command>
  <command> (%VSIM%) -c {%VSIMARGS%} -sv_seed {%seed%}
  +OVM_TESTNAME={%ITERATION%}
  +OVM_VERBOSITY_LEVEL={%OVM_VERBOSITY_LEVEL%}
  +PATTERNSET_FILENAME={%PATTERNSET_FILENAME%}
  +PATTERNSET_MAXCOUNT={%PATTERNSET_MAXCOUNT%} optimized</command>
  <command> } else {</command>
  <command> echo "Error: Cannot find patternset
  file:{%PATTERNSET_FILENAME%}, for test:{%ITERATION%}. Aborting
  simulation for the test!"</command>
  <command> }</command>
  <command>} else {</command>
  <command> (%VSIM%) -c {%VSIMARGS%} -sv_seed {%seed%}
  +OVM_TESTNAME={%ITERATION%}
  +OVM_VERBOSITY_LEVEL={%OVM_VERBOSITY_LEVEL%} optimized</command>
  <command>}</command>
</execScript>
</runnable>

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

<runnable name="vm_report" type="task">
  <execScript >
    <command>if ([file exists {%mergefile%}]) {vcov report -html {%mergefile%} -htmldir
[file join {%DATADIR%} cov_html_summary]}</command>

```

```
<command>if ([file exists (%trriagefile%)]) {trriage report -name (%trriagefile%) -file
[file join (%DATADIR%) RA_summary.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 == -->
<!-- == overridden 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 100 errors are generated
    during execution of a regression -->
  proc StopRunning {userdata} {
    upvar $userdata data
    set result [expr {$data(fail) == 100}]
    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 to create the
    triagedb file
    This example is for illustration purposes only and
    not recommended for real projects -->
  proc OkToTriage {userdata} {
    upvar $userdata data
    return 1 ; approve triage.
  }
</usertcl>
</rmdb>
```

Chapter 6

Analyzing Test Results Based on a Plan

This chapter assumes a basic knowledge and understanding of the Unified Coverage DataBase (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, refer to “[Coverage and Verification Management in the UCDB](#)” in the User’s Manual:

- coverage algorithms for coverage summary
- merging tests, and test-associated data information
- 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 demonstrates the use of the Verification Management tracker window 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 used to track testplans must have all this information merged into a single UCDB file, which is loaded into Coverage View (with vsim -viewcov) mode. This in-memory UCDB file is then viewed within the Verification Management Tracker window.

Note

 If the loaded, merged UCDB contains more than 100 tests, you are required to load the test-associated data to be able to use any of the test analysis features — such as ranking or otherwise analyzing test data — Command Line: [coverage loadtestassoc](#) or the GUI: **File>Load>UCDB Testassociated.**

Analyzing a plan entails the following specific actions:

Preparing the UCDB File for Tracking	156
Test Data in the Tracker Window	157
Viewing Test Data in the Tracker Window	157
Invoking Coverage View Mode	158
Changing Goal, Weight and User Attribute Values in Tracker	159
Analyzing Results with coverage analyze	160
Refreshing Tracker after Changing a Plan	161

Storing User Attributes in UCDB	161
Filtering in the Tracker Window	163
Filtering Overview	163
Filtering Data in a Tracker Window	164
The Create Filter Dialog Box	166
Add/Modify Criteria Dialog Box	168
Retrieving Test Attribute Record Content	168
Analysis for Late-stage ECO Changes	169

Preparing the UCDB File for Tracking

In order to track test data, the files containing the data should be merged with the testplan UCDB.

Chapter 2 – [Capturing and Managing a Verification Plan](#) – provides details 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, you must 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:

```
vccover 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, refer to “[About the Merge Algorithm](#)” and “[Limitations of Merge for Coverage Analysis](#)” in the User’s Manual.

For detailed instructions for merging the testplan with the test data, as well as linking between the two, refer to “[Merging Verification Plan with Test Data](#).”

Related Topics

[coverage analyze \[Questa SIM Command Reference Manual\]](#)

[Understanding Stored Test Data in the UCDB \[Questa SIM User's Manual\]](#)

[Merging Verification Plan with Test Data](#)

[Viewing Test Data in the Browser Window \[Questa SIM User's Manual\]](#)

Test Data in the Tracker Window

Test data related to the status of the verification of your design, including the coverage calculations, can be seen 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.

Viewing Test Data in the Tracker Window.....	157
Invoking Coverage View Mode	158
Changing Goal, Weight and User Attribute Values in Tracker	159
Analyzing Results with coverage analyze	160

Viewing Test Data in the Tracker Window

Test data, including the coverage calculations, is perhaps best viewed 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. Refer to “[Importing an XML Verification Plan.](#)”
- Merged your design test data with the verification plan into a single UCDB. Refer to “[Merging Verification Plan with Test Data.](#)”
- Merged file must be viewed in Coverage View mode. Refer to “[Invoking Coverage View Mode.](#)”

Procedure

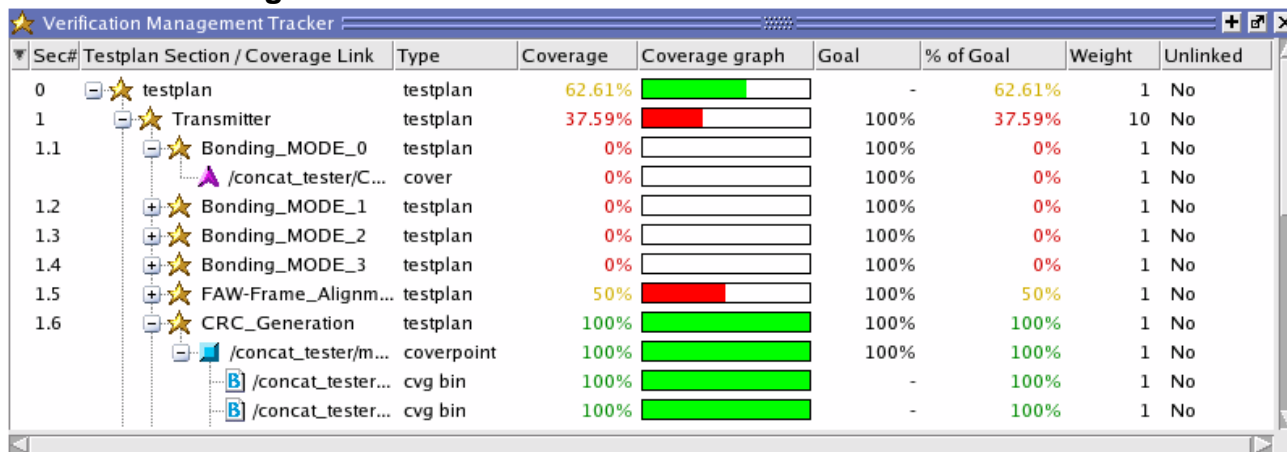
1. Open the Verification Browser window, if it is not open already:
Choose **View > Verification Management > Browser** from the main menu.
2. Double-click the merged file (UCDB) that contains verification testplan and test results. If you do not have a merged UCDB that contains a testplan, see the Questa SIM 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 choose **Invoke Coverage View Mode**.
3. The Verification Tracker window appears, similar to [Figure 6-1](#). The coverage statistics displayed in the Coverage column in the Tracker depend on whether the item is a design unit or an instance. (Refer to “[Calculation for Total Coverage](#)” in the User’s Manual for more information.) Linked covergroup, coverpoint, and cross coverage scopes are displayed, including child items of those items, to the bin level.

Figure 6-1. Test Data in Verification Tracker Window



Sec#	Testplan Section / Coverage Link	Type	Coverage	Coverage graph	Goal	% of Goal	Weight	Unlinked
0	★ testplan	testplan	62.61%	<div><div></div></div>	-	62.61%	1	No
1	★ Transmitter	testplan	37.59%	<div><div></div></div>	100%	37.59%	10	No
1.1	★ Bonding_MODE_0	testplan	0%	<div><div></div></div>	100%	0%	1	No
	✱ /concat_tester/C...	cover	0%	<div><div></div></div>	100%	0%	1	No
1.2	★ Bonding_MODE_1	testplan	0%	<div><div></div></div>	100%	0%	1	No
1.3	★ Bonding_MODE_2	testplan	0%	<div><div></div></div>	100%	0%	1	No
1.4	★ Bonding_MODE_3	testplan	0%	<div><div></div></div>	100%	0%	1	No
1.5	★ FAW-Frame_Alignm...	testplan	50%	<div><div></div></div>	100%	50%	1	No
1.6	★ CRC_Generation	testplan	100%	<div><div></div></div>	100%	100%	1	No
	✱ /concat_tester/m...	coverpoint	100%	<div><div></div></div>	100%	100%	1	No
	✱ /concat_tester...	cvg bin	100%	<div><div></div></div>	-	100%	1	No
	✱ /concat_tester...	cvg bin	100%	<div><div></div></div>	-	100%	1	No

Related Topics

[coverage analyze \[Questa SIM Command Reference Manual\]](#)

[Invoking Coverage View Mode](#)

[Understanding Stored Test Data in the UCDB \[Questa SIM User's Manual\]](#)

[Refreshing Tracker after Changing a Plan](#)

[Calculation for Total Coverage \[Questa SIM User's Manual\]](#)

[Merging Verification Plan with Test Data](#)

[Importing an XML Verification Plan](#)

[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.

From the GUI

1. Right-click to select .ucdb file. This functionality does not work on .rank files.
2. Choose **Invoke CoverageView Mode**.

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

From the 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 \[Questa SIM User's Manual\]](#)

[Viewing Test Data in the Tracker Window](#)

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 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. Press the Enter key.
4. Optionally, save the changed values by choosing **Tools > Coverage Save** or by entering the coverage save command at the command line.
5. If you exit the session without saving, a dialog box pops up asking if you want to save the changes.

Tip

i 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 \[Questa SIM User's Manual\]](#)

[Filtering in the Tracker Window](#)

Analyzing Results with coverage analyze

The coverage analyze 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.

The [coverage analyze](#) command — valid only in Coverage View mode (vsim -viewcov) — is the command on which the Tracker window is based.

The following are some sample coverage analyze commands you can use to view and analyze the verification data:

- 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.

Procedure

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 choose **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.

Use a [coverage attribute](#) command, such as:

coverage attribute -test testname -name Responsible -value Joe

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, refer to “[Understanding Stored Test Data in the UCDB](#)” in the User’s Manual.

Filtering in the Tracker Window

You can use filtering in the Tracker window to trace verification requirements.

Filtering Overview	163
Filtering Data in a Tracker Window	164
The Create Filter Dialog Box	166
Add/Modify Criteria Dialog Box	168

Filtering Overview

For example you can filter for testplan sections or linked coverage items:

- that return a specified percentage of completion.
- that match a specified weight.
- that match user attributes.

Filtering information by user attribute is the most powerful method. For example, if your original testplan included such data fields as the engineer responsible for tests or the priority level assigned to a specific section of the testplan, you can use these attributes for filtering the coverage results.

When you use the filtering functionality in the Tracker window, you are setting up rules as to what should be shown. There are some basic rules that determine when a row will be selected and shown in the Tracker window based upon your filtering criteria, they include:

- All ancestors of a row that match your criteria are shown, even if they do not match your criteria.
- When matching attributes, if a row does not have the specified attribute(s) it will not be shown. This most typically impacts the testplan rows, resulting in an empty window if your criteria are not set properly.
- Coverage total calculation is based upon the tests selected in the Specify Tests tab and further affected by your Aggregate Coverage Totals selection from the Selection Criteria tab.
- It is important to place user specified filterable testplan attributes only at the leaf level testplan items (an item which has a link to a coverage object). When you do so, the correct aggregate total coverage is shown, and it is propagated upward correctly.

Terminology:

- Row — An item in the Tracker window. This can be a testplan section or coverage item.

- Ancestor (Parent) — Any row hierarchically above a given row; a parent is directly above the row.
- Descendant (Child) — Any row hierarchically below a given row; a child is directly below the row.
- Selected row — Any row that meets the filtering criteria.

Filtering Data in a Tracker Window

You can filter data in the Tracker window in order to prune away aspects you do not need to see, or to focus on specific portions of the data.

Prerequisites

- Import your testplan (see [““Importing an XML Verification Plan” on page 45”](#)).
- Merge your testplan with test results (see [““Merging Verification Plan with Test Data” on page 56”](#)).
- Open the Verification Management Tracker window (see [“Viewing Test Data in the Tracker Window”](#)).

Procedures

This is a high level description of the filtering process. Links to detailed reference information about each sub-dialog are provided.

Create a filter

1. Right-click anywhere in the Tracker window and choose **Filter > Setup** to display the Filter Setup dialog box.
2. Choose **Create** to display the [The Create Filter Dialog Box](#).
 - a. Filter Name — Enter a name for the filter. This is what appears in the Filter dropdown after creation.
 - b. Selection Criteria tab — Create and control the criteria for the filter. Refer to the [Add/Modify Criteria Dialog Box](#) for details about creating criteria and Create Filter dialog box for additional reference information. You can create as many criteria as you need to accomplish your goal.
 - c. Specify Tests tab — Select the tests to be affected by the filter. The default is to use all tests.

Load a Filter

Right-click anywhere in the Tracker window and choose **Filter > Apply > name**, where *name* identifies the filter you have previously created.

Other Filter Operations

- Restoring all data — **Select Filter > Apply > NoFilter**.
- Filter Setup Tasks — **Select Filter > Setup**.
 - Removing a filter — Select the filter, then click **Remove**. Results in a confirmation dialog.
 - Copying a filter — Select the filter you want to copy, click **Copy**, then enter the new name in the Copy Filter dialog box.
 - Renaming a filter — Select the filter you want to rename, click **Rename**, then enter the new name in the Rename Filter dialog box.

Related Topics

[Understanding Stored Test Data in the UCDB \[Questa SIM User's Manual\]](#)

[coverage analyze \[Questa SIM Command Reference Manual\]](#)

[Storing User Attributes in the UCDB \[Questa SIM User's Manual\]](#)

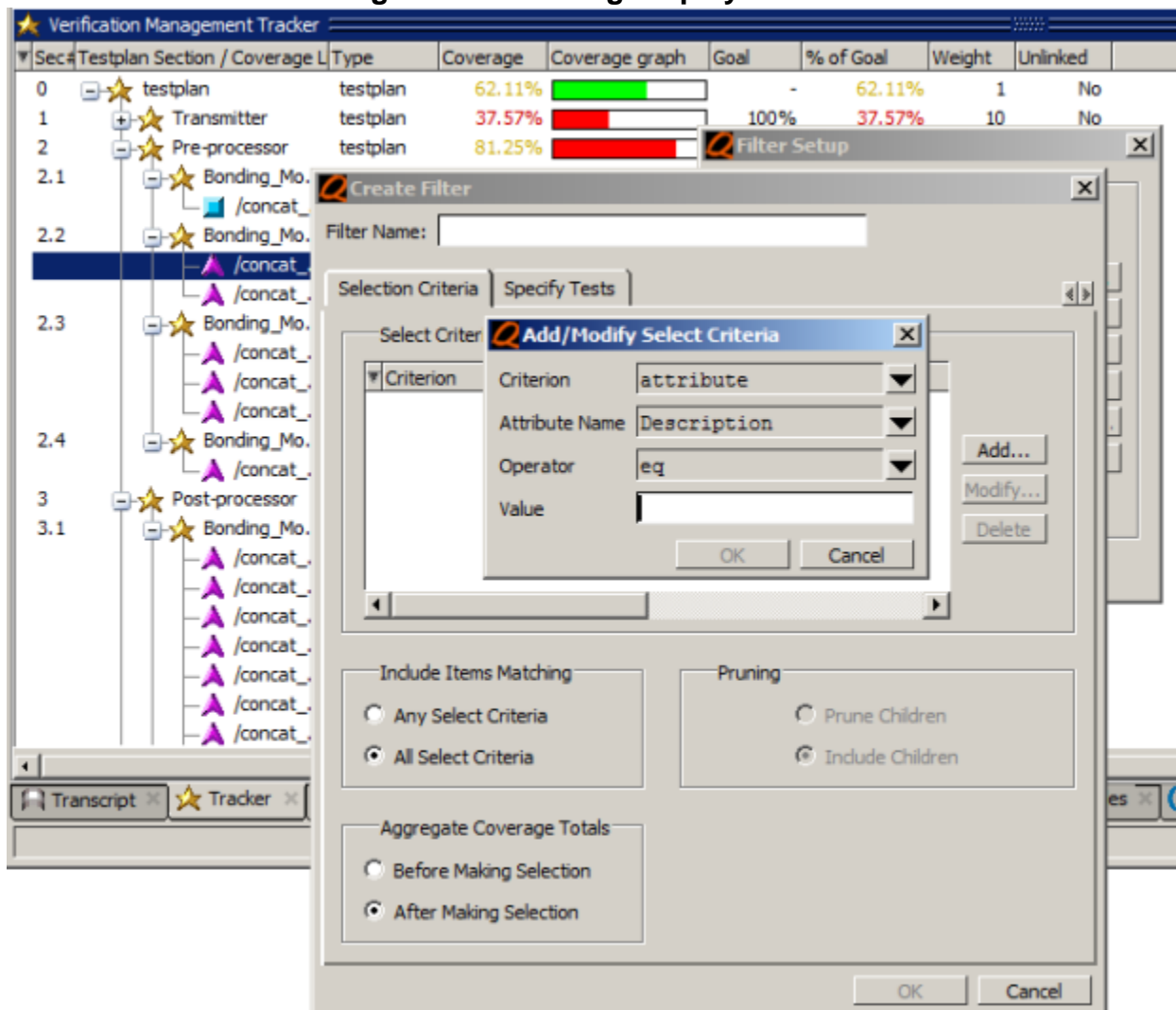
[xml2ucdb \[Questa SIM Command Reference Manual\]](#)

The Create Filter Dialog Box

The Create Filter dialog box is equivalent to the Edit Filter dialog box.

The default values match the semantics of the Contains filtering used in other tree-based windows and are recommended for regular use.

Figure 6-2. Filtering Displayed Data



Objects

- Filter Name

Use this field to enter the name of the filter you are creating.

- **Selection Criteria Tab**
 - Use this tab to set the specific criteria for your filter:
 - **Select Criteria** — Lists existing criteria and allows you to:
 - **Select Criteria** — Lists existing criteria and allows you to:
 - Add a new criterion or Modify a selected criterion. Refer to the [Add/Modify Criteria Dialog Box](#) section for detailed information
 - **Include Items Matching**
 - **Any Select Criteria** — Performs an OR operation.
 - **All Select Criteria** — Performs an AND operation.
 - **Aggregate Coverage Totals**
 - **Before Making Selection** — Coverage totals are aggregated before the filter is applied, therefore the coverage totals of all descendants are included.
 - **After Making Selection** — Coverage totals are aggregated after the filter is applied, therefore the coverage totals of only the remaining descendants are included.
 - **Pruning** — Only available when you select Before Making Selection in the Aggregate Coverage Totals section.
 - **Prune Children** — If a row is selected based on your criteria, only descendants that also match the filter criteria will be shown. This will result in some descendants being filtered out.
 - **Include Children** — If a row is selected based on your criteria, then all descendants will be shown as well. This will result in some descendants being displayed that would not otherwise fit within your criteria.
- **Specify Tests Tab**

Use this tab to select which tests are to be included in the filtering. By default, all tests are included.

Add/Modify Criteria Dialog Box

This dialog box creates your individual criteria for your filter.

Use this dialog to include items in the filter list. In other words, selected items are filtered out.

Objects

Table 6-1. Content Description of the Add/Modify Criteria Dialog Box

Criterion	Attribute Name	Operator ¹	Value	Description
Attribute	list of attribute names, pre-defined or user defined.	lt, le, gt, ge, eq, ne regexp, nregexp	integer string	returns rows that have attributes that meet the criteria
Coverage	N/A	lt, le, gt, ge, eq, ne	integer	returns rows where the coverage total meets the criteria
Instance	N/A	N/A	N/A	returns rows of type "instance"
Name	N/A	eq, ne, regexp, nregexp	string	returns rows with values in the "Testplan Section/Coverage Link" column that meet the criteria
Tag	N/A	eq, ne regexp, nregexp	integer string	returns rows associated with tags assigned with the coverage tag command
unlinked	N/A	N/A	N/A	returns rows of testplan sections that are not linked to cover items
weight	N/A	lt, le, gt, ge, eq, ne	integer	

1. Operators are based on the arguments to the [coverage analyze](#) command: lt = less than; le = less than or equal to; gt = greater than; ge = greater than or equal to; eq = equal to; ne = not equal to; regexp = regular expression match; nregexp = negative regular expression match

Retrieving Test Attribute Record Content

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

To retrieve test attribute record contents from:

- the simulation database during simulation (vsim), use the [coverage attribute](#) command.

coverage attribute

- a UCDB file during simulation, use the [vcover attribute](#) command.
vcover attribute <file>.ucdb
- a UCDB file loaded with -viewcov, use the [coverage attribute](#) command.
coverage attribute -test <testname>

The Verification Browser and Tracker windows display columns which correspond to the individual test data record contents, including user-defined name/value pairs. The pre-defined attributes that appear as columns are listed in the table “[Default \(expected\) Columns/Fields for an Imported Plan](#)” on page 24.

Refer to “[Customizing the Column Views](#)” in the GUI Reference Manual 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 (refer to “[Ranking Coverage Test Data](#)” in the User’s Manual).
- Re-run tests (refer to “[Rerunning Tests and Executing Commands](#)” in the User’s Manual).

Chapter 7

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.....	171
Use Flow	172
Data Analysis Tasks.....	173
Input Files for Results Analysis Database.....	174
Creating a Results Analysis Database (TDB)	175
Message Transformation.....	178
Triage Command Examples	181
Triage Command Related Specifics.....	184
About the .tdb Database for Results Analysis.....	186
Viewing Results Analysis Data in Text Reports	192
Triage Report Formats	193
Viewing vcover diff Results in the Results Analysis Window	194
Predefined Fields in the WLF and UCDB.....	195
Transform Rules File Format.....	199
Debugging the Transforms	203

Use Models for Results Analysis

Two predominant use models for test triage are for analyzing and processing the results from multiple and checking individual simulations: regressions and individual simulations.

- multiple run “regressions” —

One main use of Results Analysis is to analyze the results of multiple test runs, in a post-process step or within a script. After running these tests, 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.

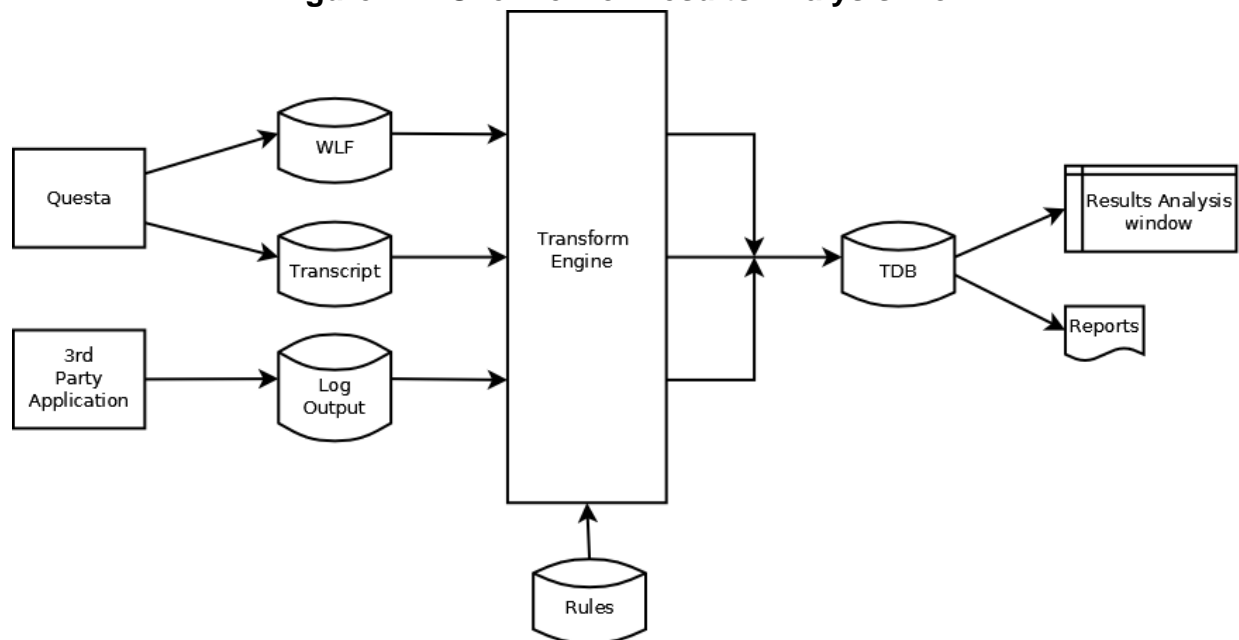
- Checking Individual Simulations —

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 generic use flow for the Results Analysis functionality is as described in this section.

Figure 7-1. Overview of Results Analysis Flow



1. Capture results to be analyzed:
 - a. 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.

- b. Merge any test results (UCDBs) you have into a single file for database creation with “[vcover merge](#)”.
2. Create the triage database (TDB):
 - a. Use either the GUI or “[triage dbfile](#)” and “[triage dbinsert](#)” commands to take message data that has been saved to a WLF file or to a plain-text LOG file, and combine the messages with the UCDB attributes from the same simulation into a triage (results analysis) database (TDB). See [Creating a Results Analysis Database \(TDB\)](#) for details.
 - b. Transform Rules File use:

If using a .wlf or UCDB file, you may want to use a transform rules file, which is a file containing one or more transform statements which are written for the purpose of improving the usability of the data that is generated in the TDB.

If using a transcript or other plain-text file, you MUST either create your own rules file or use an existing rules file to successfully extract messages. See “[Transform Rules Files](#)” and “[Example or Template Rules Files](#)” for details.
 3. View results either in reports, or by opening the .tdb you created in the Results Analysis window. The latter enables you to query results in a variety of ways, sorting by severity and other criteria. See “[Data Analysis Tasks](#)”.

Data Analysis Tasks

The basic high level tasks for analyzing Results Analysis (triage) database information are :

- Importing messages to a triage database (.tdb). Refer to “[Creating a Results Analysis Database \(TDB\)](#)”.
- Viewing the messages in the Verification Results Analysis window.

You can view the triage database (.tdb):

- From within the GUI:
 - Choose **File > Open > Results Analysis Files (*.tdb)** from the main menu.
 - Enter the triage view -name <tdb_filename>.tdb command in the Transcript window.
- From the shell prompt:
 - Enter the “`vsim <tdb_filename>.tdb`” command.

You can also view a report in text format using the [triage report](#) command.

Figure 7-2. Verification Results Analysis Window

(Default) severity -> msg -> normalize time -> 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		

Refer to “[Verification Results Analysis Window](#)” in the GUI Reference Manual, and “[Viewing Results Analysis Data in Text Reports](#)” in this manual for details on each.

- Manipulating the Results Analysis data for the purposes of analysis.

You can format the data displayed in the Results Analysis window in a number of ways to assist you in analyzing the data:

- Analysis Questions — Refer to “[Setting Custom Analysis Question Queries.](#)”
- Column Layout — Refer to “[Saving Custom Hierarchy Configuration](#)” in the GUI Reference Manual.
- Filter Expressions — Refer to “[Filtering Data in Results Analysis Database.](#)”
- Hierarchy Configurations — Refer to “[Setting Custom Hierarchy Configurations.](#)”
- Sort Configurations — Refer to “[Setting Custom Sorting Configurations.](#)”

Related Topics

[Viewing Results Analysis Data in Text Reports](#)

Input Files for Results Analysis Database

Three types of files are accepted as inputs to the Results Analysis database (TDB): UCDB, WLF, and/or LOG.

- UCDB File

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

- 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 “[Message Transformation](#)” and “[Transform Rules Files](#)” for more information.)

- 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 (that is, “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 “[Message Transformation](#)”.

- 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 rules file containing the necessary transform statement(s) in order to extract time, severity, and other types of information. See “[Transform Rules Files](#)” 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).

This involves running one or more simulations/regressions to capture results, or otherwise obtaining 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. You can also generate a results database directly from merging any test results (UCDBs) you have into a single file for database creation (.tdb) using [vcover merge](#).

Procedure

Create a triage database in either the GUI or at the command line.

From the GUI

1. Open the GUI.
2. From the main menu, choose **File > New > Results Analysis Database**

This opens the Create Results Analysis Database dialog box (Figure 7-3), which provides fields matching the switches and arguments described in the [trriage dbfile](#) command. Here, you can apply existing transform rules to the database you are creating or create new rules.

From the command line

1. Enter the [trriage dbfile](#) command:

trriage dbfile <input>

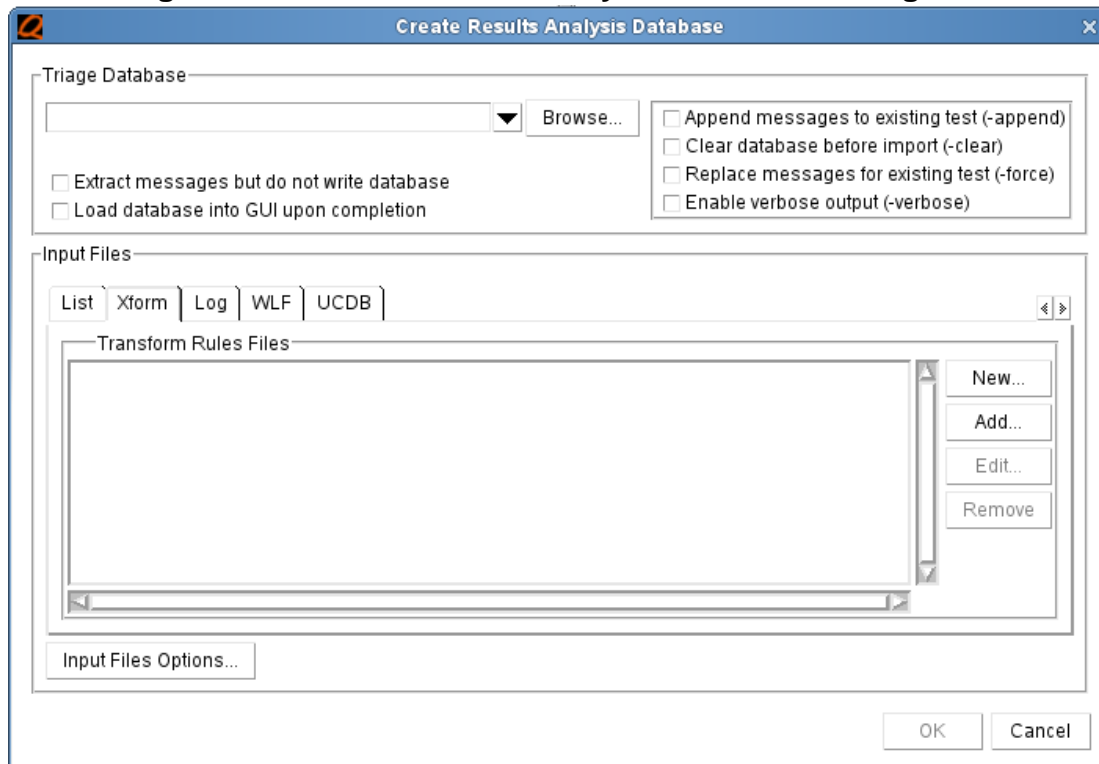
where <input> is the name of one of the following:

- a UCDB file
- a WLF file
- a plain-text LOG file

Results

Figure 7-3 shows the dialog box that allows you to configure the database for results analysis.

Figure 7-3. Create Results Analysis Database Dialog Box



Related Topics

[Input Files for Results Analysis Database](#)
[Triage Command Examples](#)
[Message Transformation](#)
[Data Analysis Tasks](#)
[Example or Template Rules Files](#)
[Filtering Data in Results Analysis Database](#)
[Transform Rules Files](#)

Message Transformation

With regards to data extraction, certain interesting and perhaps important information can 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.

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.

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).

Because transform rules are so helpful, several pre-packaged transform rules files for OVM, UVM, and so on, are available for you to use which can greatly enhance the viewing of results in the database. These rules files are located in the `<install_dir>/vm_src` directory.

Transform Rules Files	178
Example or Template Rules Files.....	179
Editing Transform Rules Files in the GUI	179

Transform Rules Files

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

- Log files require a transform rules file.
- WLF files do not require transforms, though they can be used 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. WLF and UCDB files do contain the necessary information for extraction into msg, time and severity fields, however, by default the Results Analysis tool displays all messages whose strings differ only slightly (by time, port number, or design module, and so on) 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.

Example or Template Rules Files

Several rules files containing transforms written for specific types of message transformation (OVM, UVM, and so on) have been provided for your use — as is, or as examples/templates you use to create your own rules files.

These template rules files are located in the `<install_dir>/vm_src` directory, and can assist you in creating/editing your own rules files. All transform rules files in that directory have a `.xform` extension. You will also find a `README.xform` file, which provides a brief introduction to each of the files provided and their purpose.

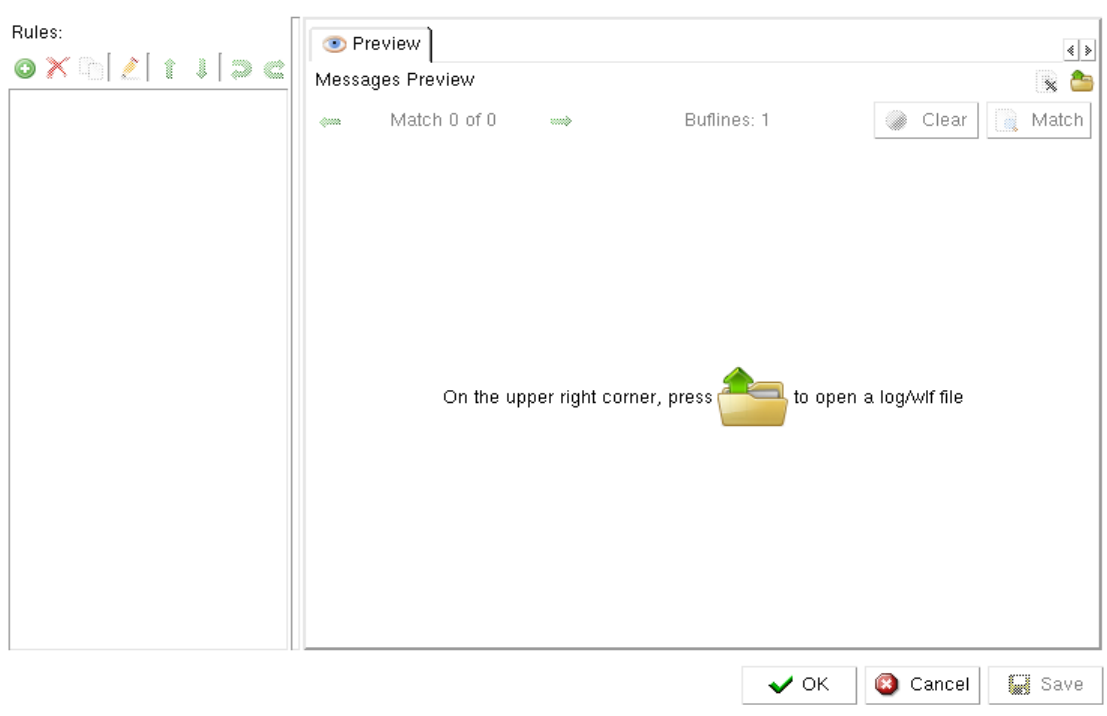
Editing Transform Rules Files in the GUI

Transform rules files can be created/edited within the GUI, through the Transform Rules File Editor dialog box.

Procedure

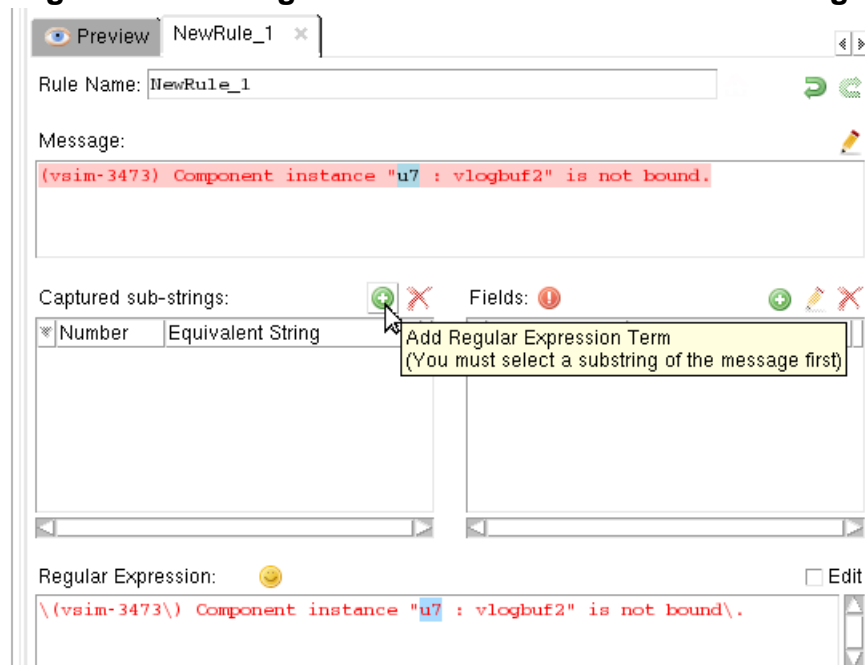
1. Open the Create Results Analysis Database dialog box by choosing:
File > New > Results Analysis Database...
2. Select the **Xform** tab.
3. To open a new Transform Rules File
 - a. Click **New**
 - b. Enter a name in the **File name** field and click **Save**.
4. To edit an existing Transform Rules File
 - a. Select a transform rules file (`.xform`) file from the list that appears
 - b. Click **Open**.
5. Step 3 or 4 opens the Transform Editor dialog box, shown in [Figure 7-5](#). You can use this editor to create, view, edit and apply transform rules to a Results Analysis database.

Figure 7-4. Transform Rule File Editor



6. Opening an existing .wlf file and creating rules from messages therein is one of the easiest ways to create new rules:
7. Click the folder icon in the right top corner of the window.
8. Open either a .log file or .wlf file from the list of files.
9. Select a message to use as a template for a new rule:
 - a. Click to select a message.
 - b. Right click and choose **Construct New Rule from Selection**.

A tab entitled "NewRule1" tab appears in the window, containing the selected message and a regular expression based on that message, as shown in [Table 7-5](#).

Figure 7-5. Editing New Transform Rule from a Message

10. You can create/edit rules, and create/edit rules files using the functions listed in [Table 7-1](#).

Table 7-1. Transform Rules File Editor Toolbar Buttons

Button	Name	Description
	Add New Rule	Add New Rule into file.
	Delete Rule	Delete highlighted rule from file.
	Clone Rule	Make a copy of this rule.
	Edit Rule	Open rule for editing.
	Move Rule (up/down)	Move the rule up or down in the file.
	Undo/Redo Change	Undo and redo changes made to file.
	Open LOG/WLF	Open a .wlf or .log file.

Related Topics

[Transform Rules Files](#)

[Example or Template Rules Files](#)

[Creating a Results Analysis Database \(TDB\)](#)

Triage Command Examples

Several triage command examples are contained in this section.

Examples

A Simple Example

The "[trriage 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:

```
trriage 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".

Over-writing Data

By default, the "[trriage 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:

```
trriage dbfile -clear example1.wlf
```

```
trriage 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:

```
trriage dbfile -clear example1.wlf
```

```
trriage dbfile -force example1.wlf
```

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 "trriage dbfile" command line.

```
trriage dbfile example1.wlf
```

```
trriage dbfile -append -rulesfile trans.xform test1.log
```

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 "trriage 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 "[trriage dbfile](#)" command:

trriage 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 "[The Testname Field and its Importance to Linking](#)".

Triage Command Related Specifics

Several issues related to the Triage command are worthy of mention:

Lockfile	184
Tip for Grid Use to Avoid Non-deterministic Results	184
Automatic WLF and Logfile Import Search Order	184

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

Search order is relevant to WLF or LOG files automatically imported as a result of having imported a UCDB file. In these cases, the following search order is observed:

1. Search using the filename as given if the filename is an absolute path
2. Search relative to the RUNCWD attribute in the UCDB file
3. Search relative to the given path of the UCDB file in the "triage dbfile" command

4. Search relative to the ORIGFILENAME attribute in the UCDB file
5. Search in the current directory

All files imported because they were specified on the command line are handled as are any other file: absolute path or relative to the current working directory.

About the .tdb Database for Results Analysis

The database (.tdb 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.

The following sections contain information related to the TDB:

The Testname Field and its Importance to Linking	186
Organizing the Results Analysis Data	187
Filtering Data in Results Analysis or Message Viewer Windows	190
Edit Filter Expression Dialog Box	190

The Testname Field and its Importance to Linking

The testname field — one of the predefined fields in the UCDB and WLF databases — is critical to linking message and test record information. It associates the data with the UCDB and WLF. The value in the testname field must match between these two databases. If the data does not match, then the two data subsets will not be linked when viewing the triage database.

Messages whose testname is blank (because the message was 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 (that is, not through 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 \(TDB\)](#)

[Data Analysis Tasks](#)

Organizing the Results Analysis Data

Key to analyzing and interpreting data the displayed in the Results Analysis database (.tdb) is how the data is configured, sorted, queried, and filtered.

The data configurations which are set using the Hierarchy Configurations, Sort Configurations, and Filter Expression are persistent: after being applied, their settings are saved for the next time you open that database. However, the Analysis Question settings are not saved from one session to the next.

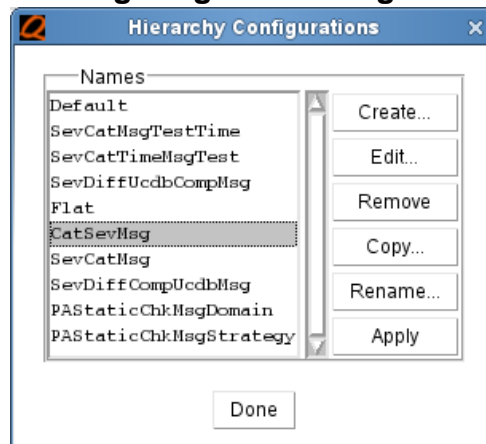
Setting Custom Hierarchy Configurations	187
Setting Custom Sorting Configurations.....	188
Setting Custom Analysis Question Queries.....	188
Filtering Data in Results Analysis Database	189

Setting Custom Hierarchy Configurations

One way to manipulate the data is through the configuration of the data hierarchy. Configurations can be managed from an open Results Analysis database, by choosing **Results Analysis > Hierarchy Configurations**.

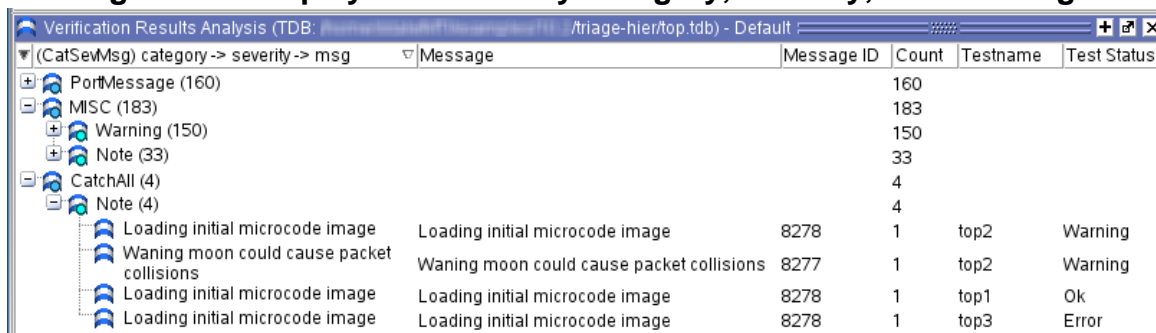
This selection opens the Hierarchy Configurations dialog. Use the dialog to create custom configurations, or choose from a selection of pre-configured options.

Figure 7-6. Configuring the Message Data Hierarchy



One such configuration is **CatSevMsg**, which presents the data by Category of the message, then by Severity, then the messages themselves, as shown in [Figure 7-7](#).

Figure 7-7. Display of Results by Category, Severity, then Message



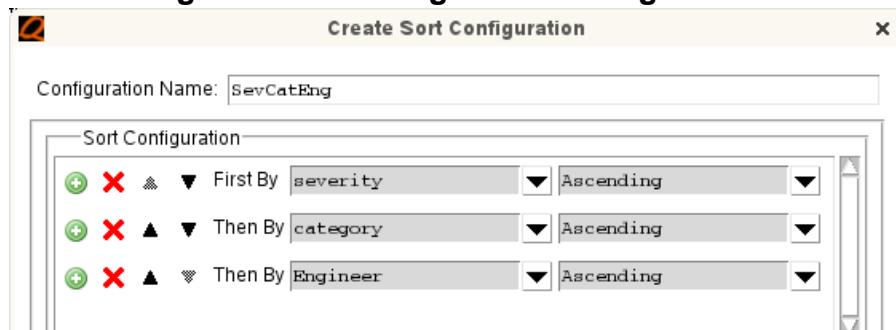
Message	Message ID	Count	Testname	Test Status
PortMessage (160)		160		
MISC (183)		183		
Warning (150)		150		
Note (33)		33		
CatchAll (4)		4		
Note (4)		4		
Loading initial microcode image	8278	1	top2	Warning
Waning moon could cause packet collisions	8277	1	top2	Warning
Loading initial microcode image	8278	1	top1	Ok
Loading initial microcode image	8278	1	top3	Error

Setting Custom Sorting Configurations

Sorting the data offers you the ability to display results analysis data by user-defined fields in a UCDB, or by multiple columns in a Results Analysis database (.tdb).

For example, in the GUI, you could create a sort configuration (**Results Analysis > Sort Configurations > [Configure Sorting...] > Create**) which organizes the data, presenting it first by Severity, then by Category of message, then by Engineer responsible for the test. (“Engineer” is an attribute previously defined in the UCDB testplan.)

Figure 7-8. Creating a Sort Configuration



Once a sort configuration has been defined in the GUI, you apply that configuration when you open a new Results Analysis window using a command such as:

```
trriage view -sortconfig <sort_config_name>
```

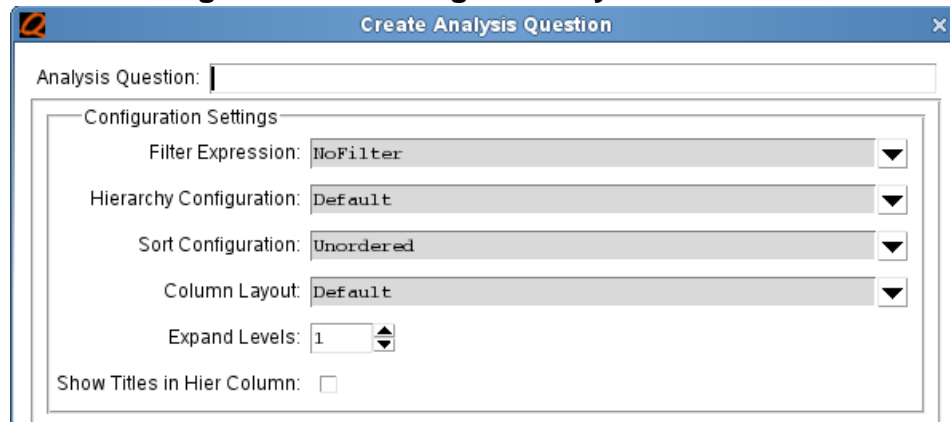
Setting Custom Analysis Question Queries

The Analysis Questions combine the application of set filters, sorts, and/or hierarchical configurations for the purpose of getting answers some important questions concerning the data collected by tests which have been run.

For example, you may want to see a list of all errors that occurred over time, or a list of the earliest error detected for each test. Both of these queries are included in a set of pre-defined Analysis Questions which are available in the drop down list.

You can also create your own queries (**Results Analysis > Analysis Questions > [Configure Question...] > Create**).

Figure 7-9. Creating an Analysis Question



Filtering Data in 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.

Filtering Data in Results Analysis or Message Viewer Windows

You can filter messages in an open database via the GUI using either the Results Analysis or Message Viewer window.

Prerequisites

You must have an open:

- .tdb (Results Analysis Database) — Refer to “[Verification Results Analysis Window](#)” in the GUI Reference Manual for details.
- .wlf (Message Viewer) using — Refer to “[Message Viewer Window](#)” in the GUI Reference Manual for details.

Procedure

1. Choose **Filter Expressions**
2. Select any of the filters listed, or create your own using > **[Configure Filter...]**
3. This opens the Edit Filter Expressions dialog box, where you can create and edit filters for the data.

Edit Filter Expression Dialog Box

Use the Editing Filter Expression dialog box to set expressions to be visible in the Results Analysis (or Message Viewer) window.

Figure 7-10. Edit Filter Expression Dialog Box

Filter Expression Terms

- severity Contains (Case Insensitive) error
- AND Choose a field... Choose a comparison operator...

First Message Filter

- ☒ Display all matching messages
- ☐ Display the first matching message from each testname, sorted by normalizetime

Time Range

- ☒ All
- ☐ Range (times in ns unless otherwise specified)
 - From
 - To

Displayed Objects

- ☒ All Objects
- ☐ Objects in the List

Add... Remove

Restore Default Filter

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 or output to a file.

Creation of the Triage Report	192
Filtering the Text Report Output	192
Counts and Compression in the Triage Report	192

Creation of the Triage Report

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.

Figure 7-11. 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> 40    Note      MISC       1 ns
#
```

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.

Counts and Compression in the Triage 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 "[Triage 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.

Triage Report Formats

There are two sets of data — detailed and non-detailed — but three different report formats that can be used.

- Non-detailed Concise

A row-and-column format showing message, count, severity, category, time, and possibly testname and ucdbfilename 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 ucdbfilename field data 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 (for example, 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.

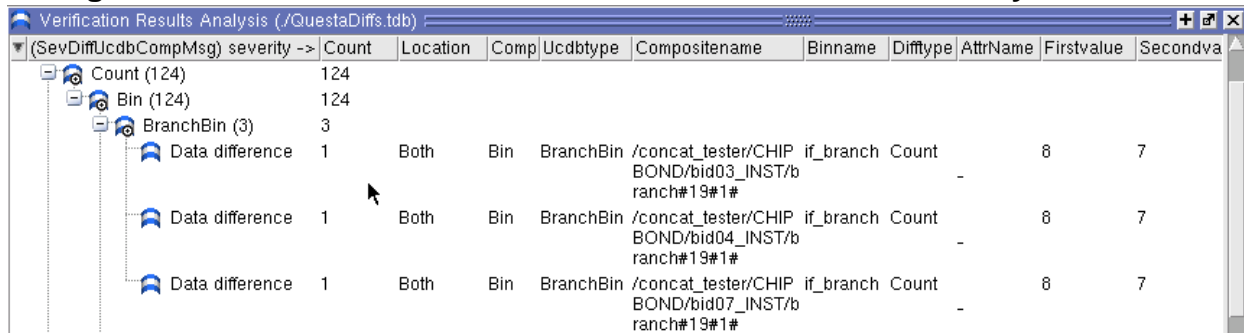
The non-detail concise and no-truncation formats both show the same data but presented in different formats. The detailed format displays extra data not found in the non-detail formats.

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 7-12. vcover diff Results in the Verification Results Analysis Window



(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	if_branch	Count		8	7
					BOND/bid03_INST/b			-		
					ranch#19#1#					
Data difference	1	Both	Bin	BranchBin	/concat_tester/CHIP	if_branch	Count		8	7
					BOND/bid04_INST/b			-		
					ranch#19#1#					
Data difference	1	Both	Bin	BranchBin	/concat_tester/CHIP	if_branch	Count		8	7
					BOND/bid07_INST/b			-		
					ranch#19#1#					

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, that is, the match basis.
- Aspect of difference (scope, bin, attribute, goal, and so on)
- 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

Predefined fields exist for both the WLF and UCDB files. It is important not to override the values of these fields.

Overview of Predefined Fields	195
Message Table Predefined Fields	196
Test Table Predefined Fields	197
User-defined Fields	198

Overview of Predefined Fields

Two major sets of predefined fields exist: one set corresponds with WLF data, the other set with UCDB Test record data.

The Message field corresponds to WLF data and the Test field corresponds to the UCDB test record data.

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

This section contains a list of predefined fields from WLF files.

[Table 7-2](#) shows these fields. The "testname" field is the primary key between this set of data and the test record information (UCDB predefined fields data).

Table 7-2. Predefined Fields from Message Table

Index #	Field
0	primaryid
1	msgid
2	msg
3	idname
4	time
5	iteration
6	category
7	severity
8	objnames
9	process
10	region
11	ucdbfilename
12	lineno
13	asrtfilename
14	asrtlineno
15	tchkkind
16	asrtstime
17	asrtexpr
18	asrtname
19	fileno
20	verbosity
21	effectivetime
23	testname

Table 7-2. Predefined Fields from Message Table (cont.)

Index #	Field
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 [trriage 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 [trriage query](#) command.

Table 7-3. Predefined Fields from Test Table

Index #	Field
0	ucdbfilename
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

Table 7-3. Predefined Fields from Test Table (cont.)

Index #	Field
16	wlfname
17	logname

User-defined Fields

Any field specified in a rules file 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 Rules File Format

In order to describe the transformation of message information from an ASCII logfile or WLF file, a rules file is required.

Syntax of transform and field Constructs	199
Transform Examples	200
Matching Messages Spanning Multiple Lines	201

Syntax of transform and field Constructs

This ASCII rules file contains the "transform" construct for message matching and a "field" construct for data extraction. The rules file supports the use of basic TCL commands and variables.

The transformation rules file, for example, *transformation.xform*, 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>] [-atleast <int>] [-nomatch <key-value-list>]
    {<field_construct>}
    [<field_construct>]
}
```

where:

-atleast <int> — Activates the nomatch body if the transform matched less than <int> time.

-nomatch <key-value-list> — Specifies the fields of the message emitted when the transform never matches any of the input messages. Four default nomatch fields are automatically set by the transform to facilitate debugging:

- transRule — transform regular expression
- transCount — the number of times the transform matched an input message
- transStLeast — the value of the atleast attribute of the transform
- transDesc — a description of the transform in the form of: No match of transform 'nameStr' (filename:lineno)"

Note



In case the key-value list does not contain a key message, the transDesc is used as the message text.

The <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 rules file, 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 its 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 rules file is required to import a plain-text LOG file.

Transform Examples

The following is a example of one complete rule within a rules file.

Transform and Field Example

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


Given a logfile with this message:

```
assert warning at time 50
```

The output of the triage dbfile command when -verbose is used 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 "wftimeunit" 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.

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

[Message Transformation](#)

[Example or Template Rules Files](#)

[Debugging the Transforms](#)

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.

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 rules file into the "triage dbfile" command, the output is:

```
# 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
```

Debugging the Transforms

Related to the debugging of transform rules, several issues should be understood:

Levels of Debugging	203
LOG File Transforms	204
The Transform Buffer and Buffer Management	204
Managing Multi-line Messages	205
Buffer Flow	205

Levels of Debugging

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.

These levels of debugging are:

- 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 rules file. 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

LOG file transforms are nearly identical to the WLF file transformations with one minor difference.

The examples shown in the section above, [Debugging the Transforms](#), are from WLF file transformations, however LOG file (that is, 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

The transform buffer and the management of this buffer are explained in this section.

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 rules file. 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. Ensure that lines following any "`\n`" occurrences in your code are intended as continuations of the message which they follow.

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 rules file.

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.

Chapter 8

Analyzing Trends

Trends in the coverage achieved over a period of time can be identified and analyzed using a Verification Management's Trend Analysis functionality. Trend analysis offers you the ability to track the progress of the coverage over time of all the objects in your verification plan.

Overview and Use Flow of Trend Analysis	208
What is a Trend UCDB?	208
Use Flow	208
Objects Available for Trend Analysis	209
Coverage Computation for Trend Analysis	209
Creating a Trend UCDB	210
Use Scenario for Adding Attributes to a Trend UCDB	211
Creating a Trend Report	212
Viewing Trend Report Data in the Trender Window	213
Trending Usage FAQs	214

Overview and Use Flow of Trend Analysis

To analyze trends in your coverage, you run tests, create the necessary trend UCDB, and create a trend report.

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.

What is a Trend UCDB?	208
Use Flow	208
Objects Available for Trend Analysis	209
Coverage Computation for Trend Analysis	209

What is a Trend UCDB?

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.

A trend UCDB database differs from a standard UCDB in that a trend UCDB:

- captures trendable objects only - that is, only down to the coverpoint or cross level.
- preserves all distinct coverage numbers for those objects, which are based on timestamp of the input. Redundant data is discarded, as it is detected due to this timestamping.

Use Flow

The use flow for trending can be summarized as follows.

1. Run regressions over multiple days/weeks.
2. Create a trend UCDB of those test runs, which is a merged database from all these regressions.

Once a trend UCDB exists, it appears in the **Verification Management > Browser > Trend Analysis** in the GUI.

3. From the trend UCDB you create a trend report, which analyzes the coverage trends of those tests over time.

The trend report includes a 2-dimensional graph, available for viewing in:

- the GUI in the Trender pane
- HTML report

- XML report
- text format
- an exported format such as CSV

Objects Available for Trend Analysis

The following is 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

Related Topics

[Creating a Trend UCDB](#)

[Creating a Trend Report](#)

[Coverage Computation for Trend Analysis](#)

[Trending Usage FAQs](#)

Coverage Computation for Trend Analysis

Trend analysis figures are based on algorithms consistent with the Questa SIM when calculating coverage, as described in this section.

Coverage computation follows the algorithms listed in “[Calculation of Total Coverage](#)” in the User’s Manual, so 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.

Related Topics

[What is a Trend UCDB?](#)

[Creating a Trend UCDB](#)

[Objects Available for Trend Analysis](#)

Creating a Trend UCDB

You can create a Trend database (a specially purposed UCDB) using the command line or the GUI. The input UCDBs need not be merged files; they can be individual files. They can also be trend UCDBs themselves.

Prerequisites

You must have previously created UCDBs, from test runs for which you want to capture trending information.

Procedure


1. From 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

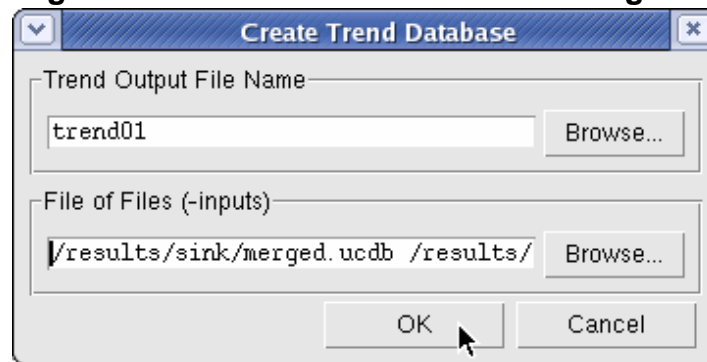
 The external coverage utility, vcover, is available in both operating system shells and the vsim command (Tcl) shell.

2. From GUI:

To create the database that can be opened in the Trender window:

3. View the UCDB(s) the **Verification Management > Browser**.
4. Select the UCDB(s) to be merged into a Trend UCDB, and right click to bring up the menu.
5. Choose **Trend Analysis > Create Trend Database**

The Create Trend Database dialog box opens up in a tab, as shown in [Figure 8-1](#).

Figure 8-1. Create Trend Database Dialog Box

6. 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.
7. 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.
8. Click **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
```

These attributes, successfully added, are now viewable using “vcover report -trend”.

Related Topics

[Objects Available for Trend Analysis](#)

Use Scenario for Adding Attributes to a Trend UCDB

You can add your own attributes to a Trend UCDB for the purpose of seeing that name/value expressed in the Trend report.

One potential usage scenario for adding attributes (test, fails, passes) for trending, over a period of time, would be as follows.

1. Run tests. Assume that 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
```

4. These added attributes are ultimately viewable using `vcover report -trend`.

Related Topics

[Creating a Trend Report](#)

Creating a Trend Report

You can create a report for viewing trends in coverage in HTML, XML, Text, or Comma Separated Value (CSV) files.

Prerequisites

You must have already created a trend UCDB file. The Trend report is created from the trend UCDB.

Procedure

Use the `vcover report -trend` command to create the trend report.

Results

The resulting report contains a 2-dimensional graph. See [vcover report](#) for details on syntax and usage.

Viewing Trend Report Data in the Trender Window

The Trend report data can be viewed in the Trender window in the Browser, after following a short sequence of steps.

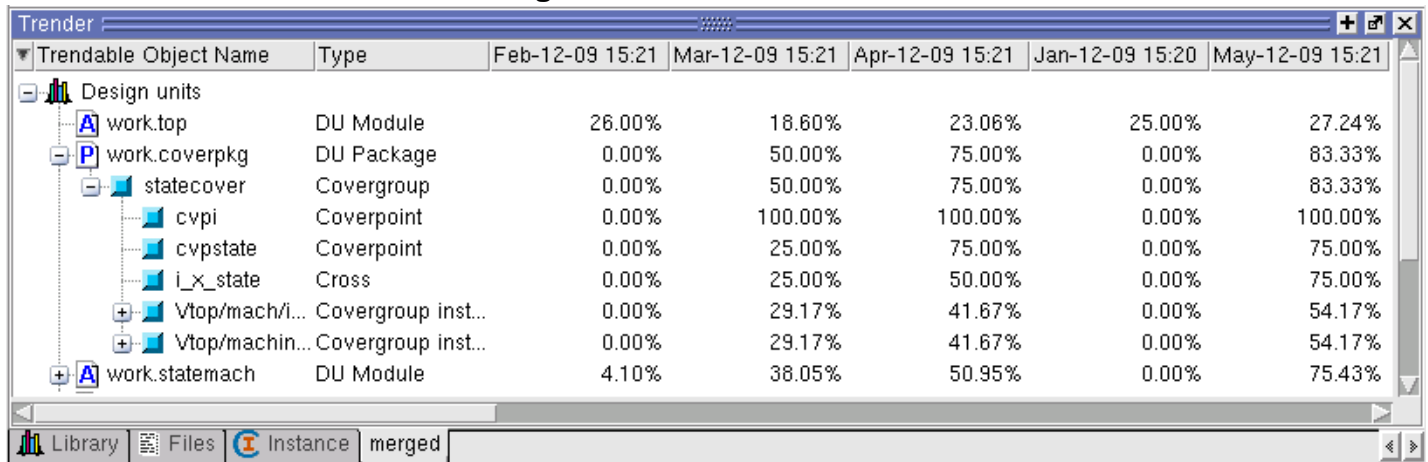
(Refer to [Verification Trender Window](#) in the GUI Reference Manual for a description of the window.)

Procedure

1. View the trend database by choosing **Verification Management > Browser**.
2. Select a trend UCDB to analyze, and right click to bring up the menu.
3. Choose **Trend Analysis > View Trender**

The Trender window opens up in a tab, as shown in [Figure 8-2](#).

Figure 8-2. Trender Window



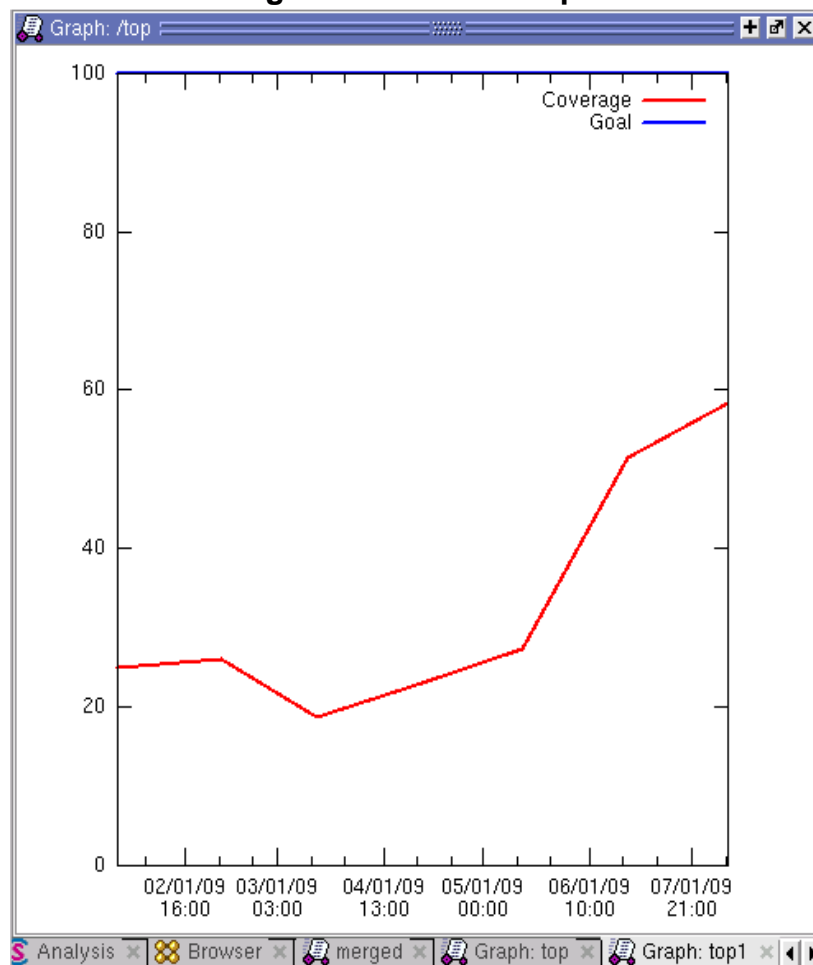
Trendable Object Name	Type	Feb-12-09 15:21	Mar-12-09 15:21	Apr-12-09 15:21	Jan-12-09 15:20	May-12-09 15:21
Design units						
work.top	DU Module	26.00%	18.60%	23.06%	25.00%	27.24%
work.coverpkg	DU Package	0.00%	50.00%	75.00%	0.00%	83.33%
statecover	Covergroup	0.00%	50.00%	75.00%	0.00%	83.33%
cvpi	Coverpoint	0.00%	100.00%	100.00%	0.00%	100.00%
cvpstate	Coverpoint	0.00%	25.00%	75.00%	0.00%	75.00%
i_x_state	Cross	0.00%	25.00%	50.00%	0.00%	75.00%
Vtop/mach/...	Covergroup inst...	0.00%	29.17%	41.67%	0.00%	54.17%
Vtop/machin...	Covergroup inst...	0.00%	29.17%	41.67%	0.00%	54.17%
work.statemach	DU Module	4.10%	38.05%	50.95%	0.00%	75.43%

4. Open the graph from the Trender window:
 - a. Select the objects whose coverage over time you wish to examine. Multiple objects can be tracked in a single window.

- b. Right click and choose one of the following from the popup menu:
- **View Trend Graph Using Local Coverage** — opens graph with local aggregation.
 - **View Trend Graph Using Recursive Coverage** — opens graph with recursive aggregation.
 - **Select Coverage and View Graph...** — opens a dialog box for enabling coverage types and recursive aggregation for viewing in the graph.

Choosing any of these displays a two-dimensional graph in the Verification Management area, similar to that shown in [Figure 8-3](#).

Figure 8-3. Trend Graph



Trending Usage FAQs

The following frequently asked questions are offered in this section to assist you during the process of trending.

Q: Is it faster to create the trend UCDB from a merged UCDB, or from a list of individual simulation UCDBs?

A: It does not matter, either way is just as effective. Timestamp totals are read from a UCDB in both cases, it does not matter if the UCDB contains 1 or multiple tests.

Q: What is the recommended method of generating trend UCDB? Let's say I run a regression each week. Do I generate a trend UCDB each week and keep only that file? Or would I keep around the merged UCDB from each week and create a trend UCDB from the merged files as needed?

A: The system supports whatever you want to do, however, there are certain advantages to each method.

Normally a trend snapshot would be added to a trend UCDB each and every time a regression is run. Trending, just like test-association merging is about the intelligent reduction of data. A trend database only keeps DU totals, instance totals, testplan totals, and functional coverage totals.

Now, if you want access to more of the weekly and monthly data, then you would need to set up a system that saves data with a window in time. This would mean that data from weeks back would only be available as a trend, and data in the window would still keep regression merge UCDBs around for further analysis, with maybe the last couple of regressions having all the UCDBs available. This can depend on your regression system and the way you chose to manage data.

Appendix A

xml2ucdb.ini Configuration File

This appendix contains the file conventions, formatting 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.

Note



The functionality described in this chapter requires an additional license feature for ModelSim SE. Refer to "[License Feature Names](#)" in the Installation and Licensing Guide for more information, or contact your Mentor Graphics sales representative.

Overview	218
When to Modify the xml2ucdb.ini File?	218
XML Version Support	218
xml2ucdb.ini File Format	218
varfile Setting for Parameterizing Testplans	222
Parameters Customizing the Testplan Import	223
Parameters Controlling the Configuration File	223
Parameters Controlling the Inclusion of Data	224
Parameters Identifying Section or Column Boundaries	225
Parameters Mapping Testplan Data Items	226
Case Sensitivity and Field Mapping and Attribute Names	231
Parameters Determining Hierarchy and Section Numbering	232
Parameters Adding Tag-based Prefixes to Section Numbers	233
Parameters Specifying UCDB Details	234
Parameters Specifying a Pre-Process XSL Transformation	234

Overview

The xml2ucdb utility uses the configuration file to determine how to process the testplan being converted into a UCDB.

The following issues are relevant to its purpose and use:

When to Modify the xml2ucdb.ini File? 218

XML Version Support. 218

When to Modify the xml2ucdb.ini File?

An *xml2ucdb.ini* file is shipped with Questa SIM which is sufficient for configuring the vast majority of all testplan imports. Generally speaking, you should not need to modify this file to import a verification plan so long as the verification plan (a top level testplan) was created:

- with the Excel or Calc extensions
- using one of the supported formats (see [Supported Plan Formats](#))
- following the guidelines listed for that particular format (see [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. Customization is achieved through modifications to this *xml2ucdb.ini* file. The format and syntax details provided in this chapter are to assist you in customizing them for your use.

Related Topics

[Parameters Customizing the Testplan Import](#)

[xml2ucdb.ini File Format](#)

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.

xml2ucdb.ini File Format

The file is located in *<installDir>/vm_src/* directory.

The *xml2ucdb.ini* configuration file is where alternate settings can be specified which govern the conversion of a testplan from any supported testplan format to the UCDB format.

The xml2ucdb.ini file — or any *<config_name>.ini* file used in place of the default xml2ucdb.ini file — must conform to the following formatting and syntax rules:

- 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 the Configuration File”](#) 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 Syntax

```
{<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.

Parameters Listed by Type

The categories for each type of parameter are the following; each of the available parameters is discussed in detail in these sections:

- [“Parameters Controlling the Configuration File”](#) on page 223
- [“Parameters Controlling the Inclusion of Data”](#) on page 224
- [“Parameters Identifying Section or Column Boundaries”](#) on page 225
- [“Parameters Mapping Testplan Data Items”](#) on page 226
- [“Parameters Determining Hierarchy and Section Numbering”](#) on page 232
- [“Parameters Adding Tag-based Prefixes to Section Numbers”](#) on page 233
- [“Parameters Specifying UCDB Details”](#) on page 234
- [“Parameters Specifying a Pre-Process XSL Transformation”](#) on page 234

varfile Setting for Parameterizing Testplans

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.

To specify the use of such overrides from the *.inifile*, use the varfile setting.

The variables to override a parameter can be specified 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)

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]”, and so on) in order to function (see “[Parameters Controlling the Configuration File](#)”).

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](#)

Parameters Customizing the Testplan Import

Parameters are available to configure the importation of customized testplans.

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 the Configuration File	223
Parameters Controlling the Inclusion of Data	224
Parameters Identifying Section or Column Boundaries	225
Parameters Mapping Testplan Data Items	226
Case Sensitivity and Field Mapping and Attribute Names	231
Parameters Determining Hierarchy and Section Numbering	232
Parameters Adding Tag-based Prefixes to Section Numbers	233
Parameters Specifying UCDB Details	234
Parameters Specifying a Pre-Process XSL Transformation	234

Parameters Controlling the Configuration File

The parameters in this section relate to locating the XML files, their parameters, and/or interpreting the extraction parameters themselves.

The parameters for controlling aspects of the configuration file itself are listed in [Table A-1](#)

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)
searchpath	<path_to_XML_files>	-searchpath	Specifies one or more directories in which the xml2ucdb utility searches for the specified XML file(s), within the testplan currently being processed
tagseparators	<char> [<char>]	-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, which allows you to fine-tune the extraction process.

- The “searchpath” parameter specifies one or more directories where xml2ucdb will find the specified XML file(s) for the testplan which is currently being processed. The specified XML file(s) are the hierarchically imported XML file(s) which are specified within the testplan itself.
- The “tagseparator” parameter specifies the character(s) to 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).

Related Topics

[Importing a Hierarchical Plan](#)

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).

See the parameters in this table:

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

Data items extracted from the XML file can be assigned to specific columns or fields in the testplan. The "column/field" represents a predefined data classification, such as Weight, Goal, Section number, and so forth.

The data extraction method used depends on the format of the data. Two of the 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	226
Parameters Mapping by Attribute Name	227
Parameter for Mapping by Column Sequence	227
Parameter for Mapping by Explicit Label	230

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), you 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 parameters in [Table A-4](#) 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
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.

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. See “[Fields in the Verification Plan](#)” for related usage information on testplan contents and mapping.

Restriction




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 (name) of plan section. Use of special characters is discouraged (“*?/.”). See “Quick Overview to Linking with the Coverage Model” .
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 to Exclude Testplan Sections from Coverage” and “Counting the Coverage Contributions from Unimplemented Links” for details.
LINKEXCLUSION	LinkExclusion	Used to exclude a child scope or bin of a linked coverage item: this column contains the full or relative path with respect to the linked coverage item. See also Excluding Items from a Testplan .
LINKEXCLUSIONCOMMENT	LinkExclusionComment	Used to Specifies a comment for the excluded child scope or bin item defined in LinkExclusion.

Note

 'AtLeast', 'LinkWeight', 'Unimplemented', 'LinkExclusion', and 'LinkExclusionComment' are present in the list of known names (that is, reserved keywords), but each of these need to be specifically uncommented (edited) in the xml2ucdb.ini file as the last fields in the datafields entry in order to be automatically recognized as columns by the xml2ucdb command. The exception to this rule is that 'AtLeast' is recognized for automatically for Word plans, and is added to the UCDB.

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.

Figure A-2. 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 used for this type of mapping 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 228), 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

Certain rules apply related to case sensitivity, 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: auto-numbered and non auto-numbered.

- 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.
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.

End-User License Agreement with EDA Software Supplemental Terms

Use of software (including any updates) and/or hardware is subject to the End-User License Agreement together with the Mentor Graphics EDA Software Supplement Terms. You can view and print a copy of this agreement at:

mentor.com/eula

